Creation of a Cloud-Native Application

Building and operating applications that utilize the benefits of the cloud computing distribution approach.

Leontine Van Gerven

Internship report presented as partial requirement for obtaining the master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

2023

Creation of a Cloud-Native Application: Building and operating applications that utilize the benefits of the cloud computing distribution approach.

Leontine Van Gerven
M20190419

MGI

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

CREATION OF A CLOUD-NATIVE APPLICATION

by

Leontine Van Gerven

Internship report presented as partial requirement for obtaining the master's degree in Information Management with a specialization in Information Systems and Technologies Management

Advisor: Vítor Duarte dos Santos

March 2023

ABSTRACT

VMware is a world-renowned company in the field of cloud infrastructure and digital workspace technology which supports organizations in digital transformations. VMware accelerates digital transformation for evolving IT environments by empowering clients to adopt a software-defined strategy towards their business and information technology. Previously present in the private cloud segment, the company has recently focused on developing offers related to the public cloud.

Comprehending how to devise cloud-compatible systems has become increasingly crucial in the present times. Cloud computing is rapidly evolving from a specialized technology favored by tech-savvy companies and startups to the cornerstone on which enterprise systems are constructed for future growth. To stay competitive in the current market, both big and small organizations are adopting cloud architectures and methodologies.

As a member of the technical pre-sales team, the main goal of my internship was the design, development, and deployment of a cloud native application and therefore this will be the subject of my internship report. The application is intended to interface with an existing one and demonstrates in question the possible uses of VMware's virtualization infrastructure and automation offerings. Since its official release, the application has already been presented to various existing and prospective customers and at conferences.

The purpose of this work is to provide a permanent record of my internship experience at VMware. Through this undertaking, I am able to retrospect on the professional facets of my internship experience and the competencies I gained during the journey. This work is a descriptive and theoretical reflection, methodologically oriented towards the development of a cloud-native application in the context of my internship in the system engineering team at VMware. The scientific content of the internship of the report focuses on the benefits - not limited to scalability and maintainability - to move from a monolithic architecture to microservices.

KEYWORDS

Applications; Cloud-native; Kubernetes; Microservices; Monolithic; VMware.

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
APM	Application Performance Management
App	Application
AVS	Azure VMware Solution
AWS	Amazon Web Services
CAP	Consistency, Availability and Partition Tolerance
CEO	Chief Executive Officer
CI/CD	Continuous Integration and Continuous Delivery
CLI	Command Line Interface
COVID	Corona Virus and Disease
CP	Cloud Platform
CSS	Cascading Style Sheets
CTO	Chief Technology Officer
DevOps	Software Development and IT Operations
DevSecOps	Software Development, Security, and IT Operations
DNS	Domain Name System
EASE	Elastic Application Security Edge
EKS	Amazon Elastic Container Service for Kubernetes
EMEA	Europe, Middle East and Africa
EPIC2	Execution, Passion, Integrity, Customer and Community
ESG	Environmental, Social and Governance
ETL	Extract, Transform and Load
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IT	Information Technology

JSON	JavaScript Object Notation
K8s	Kubernetes
KaaS	Kubernetes as a Service
MA	Monolithic Architecture
MEAN	MongoDB, Express, Angular and Node
MSA	Microservices Architecture
MVC	Model-View-Controller
NPM	Node Package Manager
RAM	Random-Access Memory
RBAC	Role-Based Access Control
REST	Representational State Transfer
SDDC	Software-Defined Data Centre
SDN	Software-Defined Networking
SE	Solution Engineer
SIGTERM	SIGnal TERMinate
SOA	Service-Oriented Architecture
TKG	Tanzu Kubernetes Grid
TMC	Tanzu Mission Control
UI	User Interface
UX	User Experience
VMC	VMware Cloud
VPN	Virtual Private Network

1. INTRODUCTION

1.1 Background and Problem Identification

1.1.1. Computer Software and Software Engineering

In a little over fifty years, the function of computer software has experienced a substantial transformation (Pressman, 2005). The role of software has progressed from being a specialized tool for solving problems and analyzing information to an entire industry in and of itself. This is due to significant enhancements in hardware performance, substantial modifications in computing architectures, extensive increases in memory and storage capacity, and a broad range of unusual input and output choices. These factors have contributed to the creation of more advanced and intricate computer-based systems. In modern times, software serves a dual purpose as both a product and a medium for delivering a product. As a product, it delivers the computational power embodied by computer hardware or, more broadly, by a network of computers that can be accessed by local hardware. Software delivers the most valuable product of our era, which is information. It converts personal data into a more useful form for local contexts, manages business information to enhance competitiveness, provides access to global information networks, and facilitates the acquisition of information in all its forms.

Pressman (2005) notes that in today's industrialized world, the software industry has become a dominant force in the economy and a vital component in the development of computer-based systems and products. The days of the lone programmer have been replaced by teams of software specialists who focus on specific aspects of complex applications. Consequently, software engineering has evolved from a niche practice to a legitimate engineering discipline. Mall (2018) defines software engineering as a systematic approach to developing software using cost-effective techniques and good development practices. Despite its recognition as a subject worthy of serious research and study, developing high-quality software within budget and on time remains a challenge. The purpose of software engineering is to provide a framework for building superior software.

1.1.2. Software Architecture: From Monolithic to Microservices Architecture

The design and construction of complex software systems depend heavily on their architecture, which is the underlying structure that outlines the technical and operational requirements (Gos & Zabierowski, 2020). In the past ten years, software architecture has gained significance as a crucial aspect of software engineering (Garlan, 2000). Professionals have recognized that a well-designed architecture is a critical success factor for system development and design. Significant advancements have been made in developing the technological and methodological foundations for treating architectural design as a legitimate engineering discipline. This is because practitioners have come to realize the importance of making explicit architectural choices and utilizing prior architectural designs in the creation of new products. Nevertheless, despite these advancements, the field of software architecture is still relatively young and underdeveloped. In addition to the aforementioned challenges the ever-evolving nature of technology presents new obstacles for software architecture. This study explores some of the significant trends in software architecture research and practice while also speculating on emerging trends, challenges, and goals. The responsibility of software architecture is to optimize each property of an application, including but not limited to efficiency, manageability, scalability, reliability, modifiability, and deployability (Gos & Zabierowski, 2020). As a result, selecting an appropriate architecture is crucial during the early stages of software development. Among the numerous software architecture types available, the monolithic and microservice architecture are two of the most popular (Gos & Zabierowski, 2020).

Historically, Monolithic architecture (MA) was the conventional technique employed by large corporations for software development (De Lauretis, 2019). In this approach, all functions are contained within a single application. Monolithic architectures typically comprise multiple functions (Bucchiarone, Dragoni, Dustdar, Larsen, & Mazzara, 2018). Some benefits of monolithic applications include their ease of development, testing and deployment (De Lauretis, 2019). Nonetheless, as additional functions are added to the application, the complexity of a monolithic architecture tends to increase. As a monolith structure grows, it becomes a larger and more difficult piece of software to manage and scale (Chen, Li, & Li, 2017). For example, to augment the application's computational capabilities, the entire monolith must be duplicated. This imposes a limited overhead if the number of functions is small. However, weaknesses in this system manifest when the monolith begins to evolve, causing an increase in size and number of functionalities. Once this happens, the disadvantages of MA might outweigh its advantages; for instance, if more computational power is required, scaling a large monolith will lead to greater overhead compared to scaling a smaller one. Another drawback of a large monolith is the slow-down of development, which can hinder continuous deployment due to longer start-up times. Despite its limitations, the traditional unified model for software design, based on monolithic architecture, is still widely used, with all its associated drawbacks (De Lauretis, 2019). Many of these legacy applications are difficult to manage and update, often forcing companies to acquire new software instead of developing new functionalities within their existing systems.

The software engineering community has shown a growing inclination towards cloud computing in the last few years (De Lauretis, 2019). Modern applications are increasingly being delivered and operated on cloud platforms, which have become the preferred model. The shift in infrastructure has led to the emergence of architectural styles that capitalize on the benefits offered by cloud infrastructures. Cloud computing presents fresh opportunities for efficient deployment of scalable applications, enabling enterprise applications to adapt their computing resources dynamically in response to demand (Villamizar, et al., 2015). The microservices architecture has gained relevance in recent years and provides the benefits of cloud computing (De Lauretis, 2019). By adopting this approach, applications can become more evolvable and take advantage of the benefits associated with a multi-tenant microservices architecture. Essentially, the microservices architecture is a technique for developing software applications that draws on principles and concepts from the service-oriented architecture (SOA) style. This allows applications to be structured as a collection of small, loosely coupled software services (Dragoni, et al., 2017). The microservices architecture represents a new paradigm for programming applications, which involves composing small services that each run their own processes and communicate via lightweight mechanisms (De Lauretis, 2019). It is important to note that the term micro in microservices refers to their contribution to the application, not their lines of code.

Various motivations are encouraging the implementation of architectures based on microservices. In this internship report, we will discuss the several characteristics that make microservices easier to maintain and scale. Unlike monolith applications, microservices can be independently developed by different teams and the failure of a single microservice does not commonly impact the whole system (Taibi, Lenarduzzi, & Pahl, 2017). The first significant advantage of microservices compared to monolith applications is the maintainability (De Lauretis, 2019). Decomposing a system into self-contained and deployable services enables development teams to test and modify their services independently from other teams, streamlining distributed development. Moreover, the small size of each microservice enhances code comprehensibility, thereby improving code maintainability (Taibi, Lenarduzzi, & Pahl, 2017). Scalability is a significant advantage of microservices (De Lauretis, 2019). It is important to note that although microservices are often deployed in stateless architectures, they do not inherently possess automatic scalability. However, within a microservices architecture (MSA), each microservice can be deployed on a separate server, with varying levels of performances, and can be coded in the most suitable development language (Taibi, Lenarduzzi, & Pahl, 2017). In case of a bottleneck occurring in a particular microservice, it is possible to containerize it and run it simultaneously on multiple hosts, instead of deploying the system on a new, high-performance machine. The penultimate advantage of microservices discussed in this report, is the clear allocation of responsibilities among team members. According to Taibi, Lenarduzzi, & Pahl (2017), microservices can be developed by separate teams without external dependencies. This enables teams to have independent code bases, and the ability to create and maintain their own build environments and revisions. This clear separation of responsibilities allows large project teams to be split into smaller and more effective teams. Additionally, microservices can be developed with different technologies and structures, which means that only high-level and technical decisions need to be coordinated among teams. Other technical decisions can be made independently by each team. Since microservices can be developed and deployed independently, each team can easily develop, test, and deploy their services without relying on other teams. And thus, the adoption of a DevOps toolchain is supported by microservices. According to Behara (2019), a monolithic application is susceptible to crashing entirely due to a single error. Conversely, a microservice architecture is less prone to such incidents because it consists of smaller, self-contained units that can be deployed independently without affecting the entire system. However, it is essential to note that a microservice architecture is not inherently resilient to failures. With dozens of interconnected services, it is crucial to ensure that one faulty service does not disrupt the entire architecture. Hence, the developer needs to design the microservices in a fault-tolerant way so they can handle failures gracefully. Fault tolerance is the characteristic that allows a system to function correctly even when certain components fail (Sharma, 2020). The resilience of the entire system that uses microservices architecture relies on the ability of each service to withstand failures and the presence of fault tolerance and isolation in the runtime infrastructure to handle the failure of any service, given that microservices are deployed independently (Haselböck, Weinreich, & Buchgeher, 2017). Additionally, faulty microservices can also be quickly restarted (Taibi, Lenarduzzi, & Pahl, 2017).

To better understand the architecture of a cloud native application, it is important to look at the essence of a distributed cloud native application. One of the most popular ways to build these applications is to use containers. With containers, each service is an independent brick that can be reused in another application. A container is a portable and immutable medium for running code. It contains the program itself but also all its dependencies. For the application, I made use of the most popular containerization engine Docker. Once a container is built, each of its instances will work identically, regardless of the Docker-compatible host machine. It differs from the virtual machine by its lightness and speed of instantiation. However, there may be some security issues since all containers on the same host machine share its core. There are now ways to strongly limit the risks related to this and improve the separation of processes.

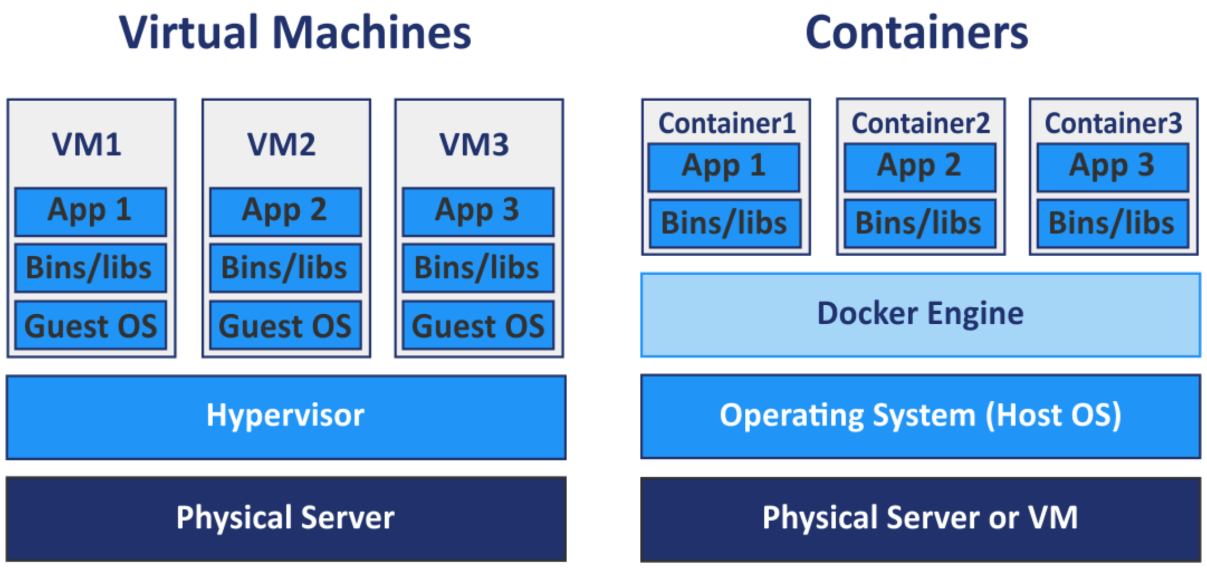


Figure 1 - Architectural Schema of VM vs Docker

But what if a container stops due to an error? How do we manage connectivity between services? To manage the entire lifecycle of a cloud native application, from storage to network, we use Kubernetes. Kubernetes is a platform that can manage containerized workloads and services, which is open-source, portable and extensible (The Kubernetes Authors, 2021). It enables both declarative configuration writing and automation. Additionally, it is a vast and rapidly expanding ecosystem, with Kubernetes services, support, and tools easily accessible. Kubernetes was first developed by Google. It is now used by all cloud providers in offerings. To name a few: the Elastic Kubernetes Service (AWS), the Google Kubernetes Engine (Google CP) or the Azure Kubernetes Service (Microsoft Azure). This type of offering is called Kubernetes as a Service (KaaS) and is gradually being adopted by all industry giants. On this platform, to join an instance of a microservice (a pod that is a member of a replica set) a "service" is used. Since each container has a unique IP, it is bound to change after a restart of the container. The service, which will have an Internet Protocol (IP) that can always be reached thanks to the Domain Name System (DNS) integrated in Kubernetes, will redirect the load to one of the instances. This way, all the microservices are accessible at any time from inside the cluster. However, since the IP of a service is not accessible from outside the cluster (e.g., from the client's browser), it is necessary to find a solution to make some services available to all. To do this, an Ingress Controller named Traefik was used. Traefik will receive all the requests thanks to an IP accessible from the outside network (routable) and will redirect them according to the URI (of type xxx/serviceA).

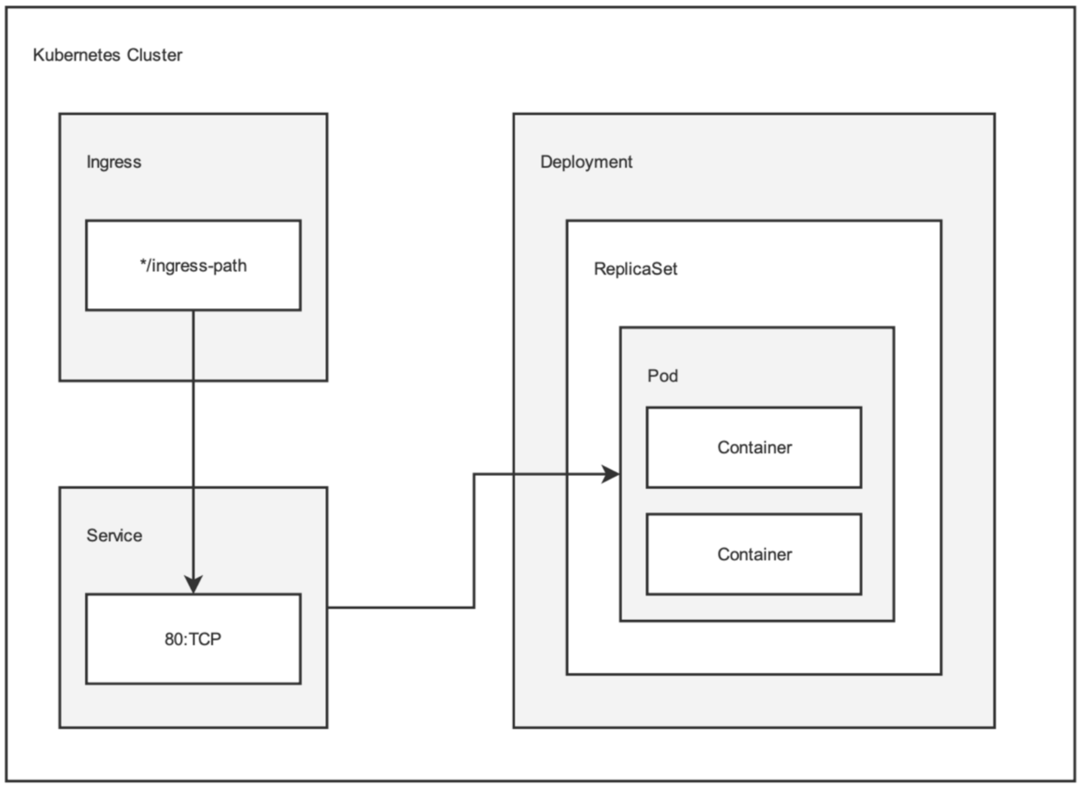


Figure 2 - Access to a Container Inside a k8s Cluster

1.1.3. The Twelve-Factor Application Methodology

With the migration of applications to the cloud, developers gain greater expertise in creating and implementing cloud-native applications (Cloud Architecture Center, 2019). The aforementioned expertise has led to the development of a collection of best practices, referred to as the twelve factors. The twelve-factor application methodology is a list of service characteristics that were defined in relation to the needs of a distributed application, intended to be deployed in the cloud. By considering these factors while designing an app, you can create apps that are highly portable and resilient when deployed to the cloud, as opposed to on-premises environments where the provisioning of new resources takes longer. In 2011, the twelve-factor methodology was developed with the aim of increasing awareness, creating a common vocabulary, and providing a comprehensive conceptual solution (Wiggins, 2017). Adam Wiggins wrote the twelve-factor app methodology, which was developed by developers at Heroku (Peltotalo, 2021). The five main characteristics of the twelve-factor app methodology, which is used for constructing software-as-a-service apps, are as follows (Wiggins, 2017):

- A **declarative** approach to automation is employed, with the aim of reducing the time and cost required for new developers joining the project.
- It is designed to be **independent** from the underlying operating system, providing the highest level of **portability** across various execution environments.
- Are well-suited for **deployment** on modern **cloud platforms**, eliminating the need for servers and systems administration.
- Aim to **minimize divergence** between development and production environments, facilitating **continuous deployment** to achieve optimal agility.

- And can be easily **scaled up** without requiring significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be utilized for applications written in any programming language, and which utilize any combination of backing services (database, queue, memory cache, etc) (Wiggins, 2017). The ultimate objective is to establish a set of guidelines that facilitate managing the expanding codebase, foster positive collaboration among developers, and prevent a decline in software quality as time progresses. Nonetheless, as technology has progressed since their initial inception, it may be necessary to modify the original guidelines in certain instances to comply with contemporary standards for application development. For instance, the methodology has been criticized for not having a general microservices approach. Thus, additional research has been done on the adaption of the twelve-factor app methodology for microservices. On a practical note, for the deployment of the application, factors that are not relevant for microservices will be used. The methodology outlines twelve factors, which are as follows:

The twelve factors	Short explanation
1. Codebase	Each service should have only one codebase, which must be tracked in revision control, and can be deployed multiple times.
2. Dependencies	Declare dependencies explicitly and isolate them from the main codebase.
3. Configuration	Configuration information should be stored in the environment.
4. Backing Services	Regard backing services as resources that are attached to the application.
5. Build, Release, Run	Ensure a clear separation between the build and run stages.
6. Processes	Run the application as one or more stateless processes.
7. Port Binding	Export services by binding them to a port.
8. Concurrency	Use the process model to scale out.
9. Disposability	Achieve maximum robustness with fast start-up and graceful shutdown.
10. Dev/Prod Parity	Strive for a high degree of similarity between the development, staging, and production environments.
11. Logs	Consider logs to be streams of events.
12. Admin Processes	Execute administrative or management tasks as single, isolated processes.

Table 1 - The twelve-factor application methodology

According to Wiggins (2017), the initial factor suggest having a single codebase for each application, from which all deployments are made. However, Horrowitz (2016) argues that, from a microservices perspective, it is better to have a separate codebase for each service. This would allow for individual deployment of each service, with deployed indicating the presence of a functional instance. As per Wiggins (2017), the second factor asserts that twelve-factor applications should not depend on the implicit presence of system-wide packages. Rather, all dependencies must be explicitly declared through a dependency declaration manifesto, both during development and production. Additionally, during execution, ad dependency isolation tool must be employed to prevent any leakage of implicit dependencies. According to the third factor, the configuration of applications should encompass all elements that are likely to differ between deployments of the codebase, such as hostnames and credentials. The main objective is to prevent the storage of any constants within the code, as twelve-factor applications require a clear separation between configuration and code. The fourth factor states that every supporting service must be regarded as a connected asset, with no differentiation in the code between local and external services. This approach ensures the highest level of adaptability and flexibility, as modifying one module does not necessitate the reconstruction of the entire application. The fifth criterion, build, release and run, requires the codebase to undergo three sequential steps to become a fully functioning release. During the release stage, the configuration should be integrated with the build to generate a comprehensive release that can ultimately be executed. Employing continuous integration and continuous delivery (CI/CD) is advisable for automating the build stage. Additionally, leveraging Docker images can facilitate the separation of the build and run stages. The sixth principle emphasizes the importance of designing application processes in a stateless and share-nothing manner. This entails that any persistent data should be stored in a stateful supporting service, while memory space or the file system can still be employed as a one-time transaction cache. Adhering to this approach simplifies scaling, as horizontal scaling can be accomplished by merely adding more services. The seventh factor entails that the web application should serve Hypertext Transfer Protocol (HTTP) as a service by binding to a designated port and handling incoming requests on that port. IN the past, when port binding was less prevalent, the webserver was inserted into the execution environment. However, a modern twelve-factor application must be entirely self-contained. The eight principle asserts that processes are treated as first-class entities. In essence, scaling out should depend on the underlying operating system, even though individual processes can still manage internal multiplexing. For example, a web process can handle HTTP requests, while worker processes can manage long-running background tasks. From a practical standpoint, containerized services can facilitate the implementation of this principle. The ninth criterion highlights the importance of making processes disposable, which implies they can be initiated or halted without delay. This feature enables services to be quickly launched, stopped or redeployed. Moreover, the shutdown process should be graceful, initiated by a SIGTERM signal from the process manager, indicating that no new requests will be accepted, while existing ones will be handled. Robustness against unexpected terminations is a crucial aspect of this factor and must be taken into account. The tenth factor underscores the importance of ensuring that all environments are as identical as feasible, while recognizing that there may still be differences in the underlying data across environments. The twelve-factor application is intended for continuous deployment, with minimal gaps in time, personnel, and tools. To achieve this, developers should create code that can be deployed hours or even minutes later, and the same individuals who wrote the code should also handle the deployment process. Additionally, consistent supporting services should be used between the development and production environments. The penultimate factor underscores the significance of logs, emphasizing that developers should not write code that is responsible for directing or storing the application's log output stream. Instead, each process should write to standard output, and one of the available logging solutions should be utilized. Logging is a critical component in troubleshooting and debugging applications, and it should be taken into account during the development process. The final criterion clarifies that administrative processes should be executed as one-off processes, distinct from the main application process. Examples of such processes include database migrations or one-time scripts. These scripts should also be committed to the application’s repository to prevent synchronization issues.

1.1.4. Introduction to the Company and its Vision

VMware is a technology company based in Palo Alto, California that specializes in cloud computing and virtualization (Wikipedia, VMware, 2021). In 1998, VMware was founded by Diane Green, a UC Berkeley graduate, her husband Mendel Rosenblum, Scott Devine, Edward Wang, and Edouard Bugnion. Today, VMware is a global leader in cloud infrastructure and digital workspace technology which supports organizations in digital transformations (VMware, About Us., 2021). VMware empowers customers to embrace a software-defined approach to business and information technology, thereby facilitating digital transformation in rapidly evolving IT environments. The company achieves this by providing cutting-edge infrastructure worldwide, catering to business of all sizes and types. Their software forms a digital foundation that powers the applications and services transforming businesses.

Figure 3 - Original Logo of the Company (VMware, 1998)

The software company initially specialized in the virtualization of the datacentre and the deployment, supervision, backup and security of these virtual infrastructures and the applications deployed on them. Virtualization consists of emulating physical objects in software. Virtualization uses software to imitate the functionality of hardware and generate a computer system that is virtual (VMware, What is Virtualization?, 2021). IT organizations are able to operate multiple virtual systems, as well as various operating systems and applications, on a single server, thanks to this. As a result, they can take advantage of economies of scale and increased efficiency. We will now briefly enumerate the four different types of virtualizations namely server, storage, network, and desktop virtualization. Server virtualization is a form of virtualization that allows several operating systems to operate as efficient virtual machines on a single physical server. This technology offers numerous advantages, such as enhanced IT productivity, lower operational expenses, faster workload deployment, improved application performance, increased server availability, and the elimination of server sprawl and complexity. The process of storage virtualization involves pooling and abstracting multiple storage devices to create a single virtual storage device (Castagna & Brown, 2021). This approach separates the software-based storage management from the underlying hardware infrastructure, making your storage resources more scalable and flexible. VMware has a storage virtualization solution called vSAN, which operates at the host level and abstracts the physical storage layer for virtual machines in a logical manner. Network virtualization provides the ability to replicate a physical network in its entirety, enabling applications to operate on a virtual network with the same functionality as a physical network, but with additional operational advantages and hardware independence (VMware, Network Virtualization, 2021). VMware NSX enables the creation of virtual networking devices and services such as logical ports, switches, routers, firewalls, load balancers, VPNs and more — for connected workloads, thereby facilitating the formation of networks comprised of virtual routers, switches, and firewalls. This type of networking is known as Software-Defined Networking or SDN. By utilizing templates, it is easier to provision environments which is beneficial for implementing Proof of Concepts (POCs) or deploying actual applications. Desktop virtualization involves the deployment of desktops as a managed service (VMware, What is Virtualization?, 2021). This approach allows IT organizations to be more responsive to evolving workplace requirements and emerging opportunities. Additionally, virtualized desktops and applications can be rapidly and easily distributed to remote locations, outsourced and offshore employees, and mobile workers using iPad and Android tablets.

VMware dominated the virtualization space. They had established themselves as a significant player in the data centre industry, but to stay a successful player in the information technology space, VMware needed to extend its strategy and embrace the cloud. During the 2010s, VMware's main focus was on establishing itself as a leader in the private cloud sector. 2016 was another pivotal moment for VMware (Mortillaro, 2018). Instead of fighting Amazon Web Services (AWS) with their public cloud offering, VMware and AWS become partners. This partnership allows to highlight the use of hybrid cloud with their hosted private cloud offer: VMware Cloud on AWS. VMware's current approach involves leveraging its extensive install base and achieve success in the multi-cloud and applications space (Wong, 2021). As companies increasingly adopt a multi-cloud strategy and embrace Kubernetes for their cloud-native modern applications, VMware is strongly pushing to expand its business. Applications serve as the backbone of modern enterprises, and thus, organizations require faster and more consistent application delivery across multi-cloud environments (Gillis, 2019). To achieve success, IT must implement automation for all aspects of their operations. This includes expanding networking and security services to cover both private and public cloud environments, which will facilitate self-service for developers. Additionally, IT should strive to gain the necessary flexibility to meet the demands of the business. It is not enough to only automate provisioning; IT must also have a complete understanding of end-user experiences and application performance from start to finish.

As such, VMware is empowering companies to succeed and thrive as a digital business. The digital economy is going full throttle. Innovative digital businesses are successful because of their use of software to invent new products and services and even create new business models (Pritchard, VMW VMware, Inc. (VMW) CEO Patrick Gelsinger on Citi's 2020 Global Technology Virtual Conference., 2020). VMware is uniquely qualified to help customers on their software-defined journey by applying the benefits of the software-defined data centre and their integrated architecture for IT to shift them to digital (Gelsinger, VMware Strategy: Value Proposition, 2016). Four critical business priorities are paired with IT initiatives to enable digital businesses to succeed. First, digital businesses must excel at creating exceptional experiences. For IT, this translates into mobilizing everything with VMware's consumer-simple, enterprise-secure digital workspace solutions. The customer can take an end-to-end approach to mobilize their business from the data centre to the desktop and the mobile device. Second, the need to differentiate the business with applications and data, which means IT must deliver cloud-native apps and services while modernizing legacy applications. VMware allows the customer to securely create, deliver and manage cloud-native and traditional apps with a common platform. It enables them to excel in the delivery of modern apps running on an agile API-driven infrastructure. In this way, the customer can make the most of mature processes and extend its current people skills and tools to embrace developer-driven operations with VMware Photon Platform. The customer can easily and securely provision and manage cloud-native, container-based applications. VMware enables the customer to deploy apps to a private or a public cloud. Next, for businesses to instantly act on new market opportunities now and in the future, IT must unify its clouds. VMware enables a unified, hybrid cloud for the power of its software-defined data centre architecture. This lets the customer serve and scale apps efficiently from the customer's choice of infrastructure. This choice of infrastructure extends to open-source options like containers and OpenStack. VMware is not only a major OpenStack contributor, but VMware also offers integrated OpenStack which makes it easy to deploy developer-friendly, enterprise-class OpenStack clouds on top of vSphere. Lastly, regardless of the underlying technology choices, preserving customer trust and safeguarding the brand are essential for any business. As a result, cybersecurity becomes one of the most critical IT imperatives. Only VMware delivers security intrinsic to infrastructure from inside the application, inside the network and at the user- and content-level. To conclude, VMware is passionate about what is possible with the software-defined approach to business.

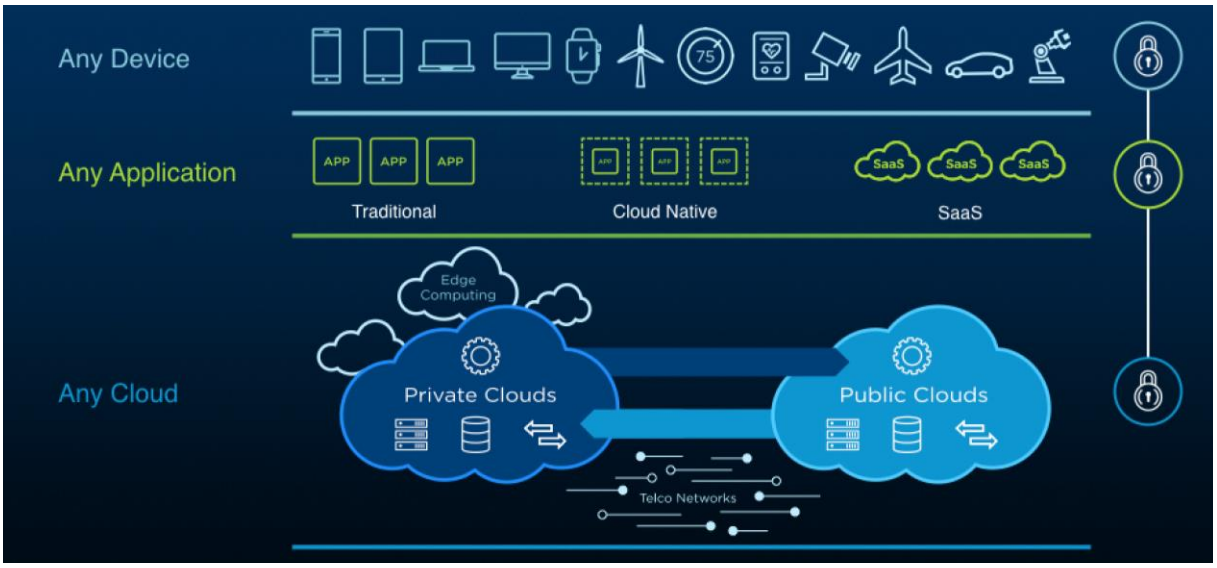


Figure 4 - VMware's Vision: The Essential, Ubiquitous Digital Foundation (VMware)

VMware aims to assist its clients in achieving digital transformation. The company's five-pronged strategy did not shift (Gelsinger, VMware Strategy: Value Proposition, 2016). It kept its focus on the same five key areas: cloud-native applications, multi-cloud, networking, security, and end-user devices. However, lately customer priorities did shift, causing some relative alterations within those five areas. Our digitally enhanced world has accelerated the distribution of people, information, and technology, delivering new and exciting opportunities for how we work and consume products, services, and experiences. But the realization of these opportunities continues to be hampered by the variety and complexity of systems, clouds, and technologies. VMware is empowering organizations to accelerate their innovation to meet the new and immersive needs of customers and employees, and to seize the opportunities in front of them today and in the future with ease, speed, and control.

In line with its current strategy and focus on multi-cloud and modern applications, VMware has in recent years acquired companies and start-ups to consolidate their positioning in this niche. At VMworld 2018, VMware has declared its acquisition of Heptio with the aim of enhancing its ability to assist business in creating and managing containerized, Kubernetes-oriented infrastructures (Lunden, 2018). Joe Beda and Craig McLuckie, two of the original creators of Kubernetes at Google in 2014, co-founded Heptio. Another company acquired by VMware in 2019 is Bitnami. With this acquisition, VMware can offer over 130 commonly used software packages in multiple formats, including virtual machines or Docker containers (Miller, 2019). This approach was very attractive to VMware as it was making its transformation to become more of a cloud services company. Finally, still in a move consistent with its goals for public cloud and cloud native apps, in December 2019, VMware obtained Pivotal, which provides a robust range of developer resources through its top-notch platform and services focused on modern application development acceleration (VMware, VMware and Pivotal, 2019). By integrating Pivotal's development-focused products with VMware's Kubernetes runtime infrastructure and management utilities, the process of software delivery can be streamlined and enhanced beyond the multi-cloud infrastructure level, resulting in improved organization, stability, and security. VMware unveiled a fresh collection of products and services called VMware Tanzu during VMworld 2019, with the aim of revolutionizing how businesses construct, operate and oversee software on Kubernetes (Ross, 2019). The foundation of VMware Tanzu comprises of VMware's well-established infrastructure products, which have been further enriched by incorporating the technologies contributed by Pivotal, Heptio, Bitnami, and various other teams within VMware (Miller, 2019). These combined efforts have resulted in an entirely new assortment of products and services. Tanzu is an evolving ecosystem of products aimed at simplifying and centralizing the deployment of applications in the cloud.

VMware is currently the only player in the world that can provide the multi-cloud platform for all the applications of an organization, wherever they want them deployed. With this platform, VMware has created the trusted foundation to enable innovation and control – without compromise. At VMworld 2021, VMware announced how they enhance this trusted foundation to accelerate innovation in cloud infrastructure and management, in application development, in edge, in security and in digital workspaces. VMware has introduced novel advancements in multi-cloud technology that assist clients in a quicker and more secure transition to the cloud, hasten the process of application modernization, and facilitate the adoption of a cloud operating model, including Elastic Application Security Edge (EASE), which enables customers to blur the boundaries for applications that extend across cloud environments. And thus, the customer can build applications on the cloud of their choice – based on the specific needs of the application – with a consistent developer experience.

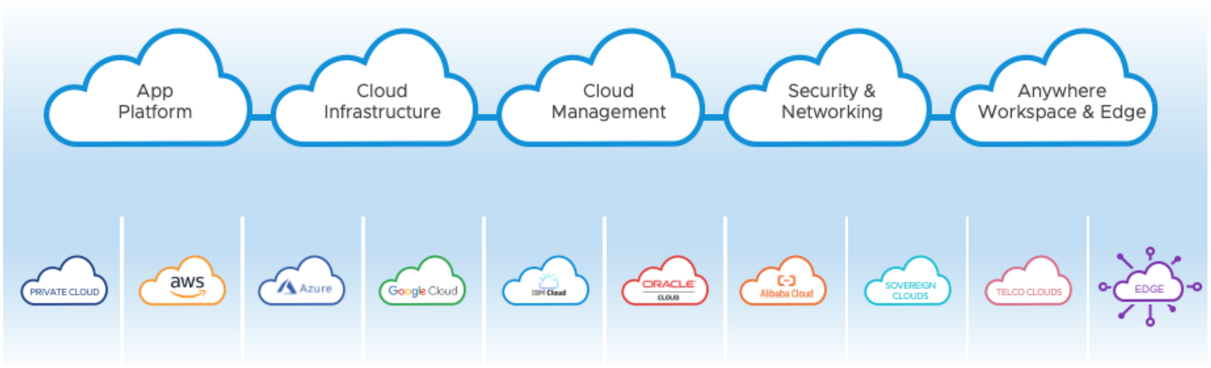


Figure 5 - VMware's Updated Vision

Virtualization and cloud computing are technologies that generate a lot of hype, but it is important to note that they are not interchangeable (VMware, What is Virtualization?, 2021). Virtualization is a type of software that decouples computing environments from their physical infrastructure, while cloud computing is a service that provides on-demand access to shared computing resources, such as software and date, over the Internet. One approach for organizations to consider is to start by virtualizing their servers and then transitioning to cloud computing for even greater self-service and agility.

1.1.5. About VMware: Recap, Highlights, and Perspective for the Future

In the course of more than twenty years, VMware has evolved from a small group of five technologists living in a small apartment to become a pioneering enterprise software innovator with a global reach (VMware, VMware Timeline, sd). Presently, VMware's digital foundation comprises multi-cloud, application, anywhere workspace, networking, and security solutions that cater the needs of more than 500,000 customers worldwide. VMware innovations fundamentally changed IT and unlocked possibilities for businesses. This section covers a chronological overview of some main events since VMware was founded and its perspective for the future.

- **1998**: The VMware story begins. Forward-thinking technologists want to create a more efficient way to compute. The company will have expanded to twenty employees by the year-end.
- **2001**: VMware presents its first product, Workstation 1.0. It enables a user to operate multiple virtual machines, each with its own operation system, on a single PC.
- **2002**: VMware launches the ESX server 1.5, its initial hypervisor. This innovative product enhances performance, simplifies IT administration, and enables organizations to save money by consolidating multiple servers onto fewer physical devices. Furthermore, VMware obtains its first patent for virtualizing computer systems.
- **2003**: VMware extends its reach globally by establishing its inaugural European office in Frimley, UK. Since then, VMware has opened numerous offices worldwide, spanning from Asia Pacific to the Middle East to Latin America.
- **2004**: The EMC Corporation purchases VMware for $635 million. This same year also marks the debut of VMworld, which would subsequently become the world's largest conference for cloud computing and virtualization.
- **2007**: VMware makes its debut as a publicly traded company on the New York Stock Exchange, with shares starting at 29 dollars and concluding the day at 51 dollars.
- **2009**: VMware introduces desktop virtualization, a new era in virtualization, with the realise of VMware's Virtual Desktop that enables delivery of desktops and applications to not just PCs and laptops, but also to mobile devices. This cloud-based deployment process streamlines IT, reduces company expenses, improves security, and provides users with remarkable mobility and flexibility. In the same year, VMware launches VMware vSphere, a virtualization platform for cloud computing that rapidly becomes a standard in the industry.
- **2012**: In the same year that Pat Gelsinger assumes the role of CEO, VMware procures the services of Nicira, a network virtualization firm whose Network Virtualization Platform facilitates the formation of virtual networks that are not reliant on physical network hardware. This acquisition proves to be a significant inclusion to VMware's software-defined data centre approach.
- **2014**: Within a mere 18 months of acquiring Nicira, VMware makes its biggest purchase yet by acquiring AirWatch. This acquisition quickly establishes VMware as a prominent contender in the Mobile Device Management industry, which enables business to accommodate the growing trend of bring your own device in the workplace.
- **2015**: VMware has earned on spot on Fortune's esteemed Best Companies to Work For list, a recognition it has consistently maintained over time. The credit goes to VMware's fundamental EPIC2 values – Execution, Passion, Integrity, Customers and Community – which form the foundation of the company's operations. In that same year, Dell Inc. declares its

intention to acquire EMC Corporation, which includes its share in VMware. This monumental purchase is the largest deal ever made in the technology sector. The acquisition brings together a distinct set of companies, operating under the name of Dell Technologies, that furnish the critical infrastructure necessary for business to construct their digital future, revamp their IT systems, and safeguard their vital information assets.

- **2016**: VMware pledges to attain carbon neutrality and attain the exclusive use of one hundred percent renewable energy for its worldwide operations by 2020, aligning with its objective of being a positive influence in the world. The VMware 2020 objectives aim to enable the company to contribute more to society, the environment, and the global economy than it consumes. In that same year, VMware, and Amazon Web Services (AWS) unveil the launch of the VMware Cloud on AWS, enabling IT teams to operate their cloud-based resources using familiar VMware tools. This collaboration marks a new age in cloud computing, where customers can operate across multiple clouds without limitations.
- **2017**: VMware unveils VMware PKS, a solution that empowers business to implement and run container services, including Kubernetes, on VMware vSphere, for production-level applications. In that same year, VMware's workforce has grown to over 20 000 employees situated around the world.
- **2019:** VMware's strategy is fast-tracked with the acquisition of two companies, namely Pivotal and Carbon Black. They enable the company to provide customers with the means to build, operate supervise, connect, and secure any application, on any cloud, and any device.
- **2021:** Raghu Raghuram takes on the role of Chief Executive Officer. In the same year, VMware separates from Dell Technologies, providing the company with enhanced autonomy to implement its multi-cloud strategy, a streamlined capital structure and governance framework, and more operational and financial maneuverability (Thacker, 2021).

Regarding the future, VMware unveiled its 2030 Agenda, which is a decade long Environmental, Social and Governance (ESG) pledge and appeal to promote a world that is more sustainable, fair, and secure (VMware, Environmental, Social and Governance., 2020). VMware's vision is to build a sustainable, equitable and secure digital foundation for all. Different business units at the company have collaborated to develop 30 cross-functional goals to meet by 2030, all of which are aligned to our three outcomes of Sustainability, Equity and Trust (VMware, The Next Normal: Our 2030 Agenda., 2021):

- VMware strives for a sustainable digital infrastructure that relies on renewable energy and operares with high efficiency, supporting the shift towards net zero emissions and decarbonization across our customers, supply chain and operations.
- Equitable, unbiased, and inclusive access to opportunities for all by enabling people to work where and how they want to work.
- VMware earns trust by staying commited to resilience, ethics, data privacy, security, and transparency.

1.2 Study Objectives

1.2.1 Introduction Solution Engineering Team

Since 1998, VMware's growth has been unprecedented (Atlantic, 2021). The fast-changing technology landscape only reinforces this trend. VMware currently employs more than 36.000 people worldwide, of which a bit more than 200 are employed in the Barcelona office, which was the place I executed my internship. Thanks to their growth, VMware is seeking to create the next generation of pre-sales Solution Engineers (SE). This aim is seen as critical to increase the adoption rate of VMware's products and services and for the company's future success more generally. VMware has industry-leading technology solutions and aims to help customers on their transformation journey (VMware, About Us., 2021). The SE team plays a role in all technical aspects of customer engagement. By working closely with sales and professional services, a solution engineer develops enduring customer relationships, ensuring the customer meets their business challenges through the adoption of VMware solutions. Solutions engineering is the ideal blend of business and technology, helping customers to understand how they can use technological innovation to transform their IT landscape and achieve their business goals. A solution engineer thus helps match the right VMware IT solutions with customer specific needs and issues to help them improve their organization in specific business units or across the board. To accomplish this, a technical sales lifecycle that consists of different steps is adopted. First, the business drivers and critical use cases needs to be understood. Secondly, the SE engages with customers to validate opportunities and then identifies best solutions and services for key use. The SE can be a specialist or a generalist. A specialist focuses on one specific product of the VMware portfolio or a product group. A generalist knows the entire portfolio on a high-level and is called a core SE. As technology is always evolving, staying up to date with trends in information technology is key for being a trusted advisor towards the client. To better address this, VMware has vastly increased its investments in training Solution Engineers and created an EMEA Academy for that purpose. This training introduces graduates to VMware and its technologies and is followed by an in-role training as part of the solution engineering team.

1.2.2 Purpose of the Internship

At the time of my internship, VMware was developing a showroom to showcase the value proposition of the software defined datacentre (SDDC). As an SE intern, I was a critical part in the evolvement of this showroom, more specifically into the development of a demo application. The demo application was an application build around a fictional customer use-case. The application was originally deployed as a legacy application on the SDDC. With the shift in VMware's strategy, my main responsibility was to extend this application to the cloud, and thus develop a cloud-native application. My internship was part of VMware's development policy of the multi-cloud and cloud native apps activity, which will be the future digital transition for many companies. The purpose of the project I worked on was to expand the possibilities of demonstrating cloud-native applications running on a VMware infrastructure and to integrate these features into an already existing demo application named Tito. The original application was a legacy application built on VMware's SDDC. It is a completely fictional customer case. The purpose of the application was to be able to rent cars. In a second version of the application, additional functionalities were added. Now, the application needed to be extended to the cloud and a procedure to automate the return of rented vehicles was added. The application can discover damages occurred to the car by analysing pictures. The confidential code for this application is stored in GitHub. When I started the project, the existing application was running on VMware Hybrid Cloud. The application was also consuming several services from external cloud providers, such as Lambda functions and relational databases from Amazon Web Services. I took over the work done by my predecessor after a handover. The application was functional in a quasi-monolithic form, which means all services were running on the same host operating system and partially deployed in Docker containers. The inter-services communications were only done through APIs, and thus in a synchronous way. This is not optimal for the operation of an application composed of micro-services distributed on the internet network. My predecessor developed a neural network using transfer learning processes, which was the heart of the business logic of the application.

The final work was intended to be presented to the whole team with a theoretical part on distributed application architectures as a preamble. The mission is to properly understand the environment and the associated technologies, to maintain the environment and to connect to the cloud. At the end of my six-months internship the application is fully functional and maintainable thanks to a well-documented source code and deployment instructions. Almost all the objectives have been achieved.

The purpose of this internship report is to share with you my experience and the work I have done during the last six months at VMware. Reading the report, you will learn more about VMware and we will go through all the different phases of application development. The focus will be on how the application is organized, the difficulties encountered, and the lessons learned. Whilst reading this report, it will become clear how this experience has been extremely beneficial in the context of my professional career, and consistent with the teaching of NOVA IMS.

2. STUDY RELEVANCE AND IMPORTANCE

2.1 OVERALL BUSINESS LANDSCAPE CHANGE

Until recently, it was the accepted practice to develop applications with the understanding that they would be installed on physical servers (Hoffman, 2016). Despite its widespread use, bare metal deployment had several drawbacks and potential risks. Scaling applications dynamically was not possible, the deployment process was complex, changes in hardware could result in application failures, and hardware failures often led to substantial data loss and extended downtime. These challenges ultimately paved the way for the virtualization revolution. The utilization of virtualization significantly reduced the cost of running business applications and necessitated only modest investments (Arora, Catlin, Forrest, & Vinter, 2020). With the advent of the hypervisor, a new layer of abstraction was introduced above the hardware, aimed at facilitating deployment, enabling horizontal scaling of applications, and reducing the likelihood of extended periods of downtime and susceptibility to hardware failures (Hoffman, 2016).

In today's interconnected world of smart devices and advanced software, virtually every company relies on software development as a crucial element (Hoffman, 2016). Software has become an indispensable part of even the most traditional manufacturing companies. The digital services an organization provides now define its identity (VMware, VMware Tanzu Intro and Vision, 2021). The importance of digital services cannot be overstated, as they provide organizations with opportunities to create new revenue streams, engage customers with novel experiences, and establish digital-first touchpoints that safeguard and empower customers and communities. However, despite high expectations, the pace of transformation is sluggish. The complexity of existing applications, interdependencies with existing infrastructure, heterogeneity across public clouds, and compartmentalization of infrastructure, operations and security have impeded progress. Nearly every organization confronts the challenge of reconciling the potential benefits of transformation with the daunting task of implementing it. Software is essential to build things efficiently and at scale, and participating in the global marketplace is impossible without it (Hoffman, 2016). The ability to deliver software that is highly reliable and can be quickly deployed is crucial for any industry to remain competitive. This software should be capable of scaling dynamically to handle previously unprecedented data volumes. Failing to create software that can effectively manage big data may result in competitors taking the lead. Additionally, if the software cannot handle massive loads, responds quickly, and adapt to market changes, competitors will likely find a way to do so.

Today, organizations that are thriving have the ability to move at an incredibly rapid pace (Buckley, 2021). This trend was also perceivable during the COVID-19 pandemic. Those companies that could get an online presence for selling, marketing, customer service, got through COVID and grew quite aggressively. Those organizations that did not adapt quickly were left behind. So, the opportunity for Tanzu – the suite of products that helps users run and manage multiple Kubernetes clusters across public and private clouds - within VMware is tremendous because VMware is taking costumers on a journey that takes those traditional applications, modernizes them, and puts them onto a very agile platform and thus the platform allows for codebases and application code to be changed extremely quick. This leads us to the fundamental concept of cloud native (Hoffman, 2016). The days when companies could tolerate being handicapped by dedicating excessive time and resources to DevOps taks, constructing and managing fragile infrastructures, and dreading the rare production deployments are long gone. In today's era of cloud computing, it is crucial to concentrate solely on one's competitive advantage and delegate non-functional requirements to platforms. Therefore, it is essential to build applications in a manner that embraces this paradigm shift.

Certainly, the adoption of cloud technology has a unique economic profile (Arora, Catlin, Forrest, & Vinter, 2020). Although leveraging the cloud necessitates investing in building capabilities and migrating applications, it is more advantageous in the long run. The most significant advantages for business include faster time-to-market, simplified innovation, easier scalability, and reduced risk. By utilizing cloud platforms, new digital customer experiences can be deployed in a matter of days instead of months. In today's world, achieving a rapid time-to-market has become imperative to avoid falling behind our competitors (Hoffman, 2016). As a result, containerization facilitates a reduced time-to-market through increased productivity, enabled by faster application deployment and portability (Nortal Cloud Team, 2017). The current competitive environment demands rapid upgrading and release of online systems or business services, making a reduced time-to-market incredibly crucial (Singleton, 2016). To capitalize on these advantages, cloud-native architectures should be adopted, and applications should be constructed under the assumption that everything is a service. (Hoffman, 2016).

2.2 THE THREE TRANSFORMATIONS OF APP AND CLOUD MODERNIZATION

The pandemic has accelerated digital business initiatives (VMware Tanzu, 2021). Organizations are undergoing a transformation in how they construct and provide innovative digital products to stay competitive (VMware, VMware Tanzu Intro and Vision, 2021). The application plays a crucial role in facilitating customer interactions and providing novel functionalities to both customers and employees. However, digital transformation takes place at multiple levels to enable these new digital initiatives and applications. Companies are currently focused on updating and creating new applications to be implemented in cloud, managed data centres, and edge locations. To aid customers in implementing their modern cloud strategy, we have identified three transformations that are expediting the provision of digital services. At VMware, we observe our clients undergoing three transformations, each of which contributes to digital business and application modernization at various stages (VMware Tanzu, 2021). The following three transformations explain how VMware goes to market.

- App Transformation
- DevSecOps Transformation
- Cloud Ops Transformation

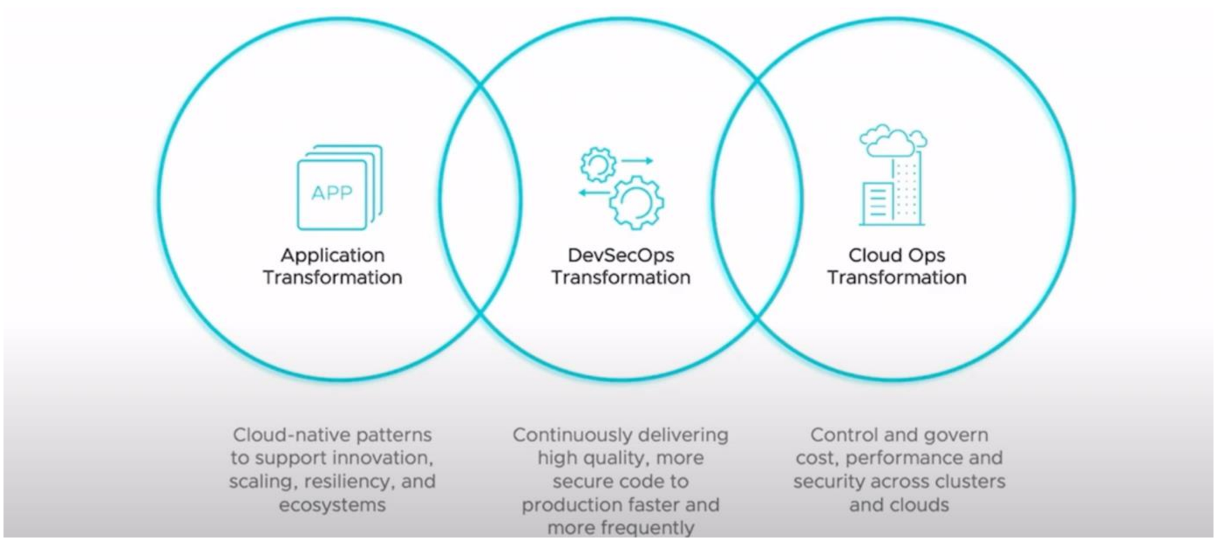

APP
Application Transformation
DevSecOps Transformation
Cloud Ops Transformation
Cloud-native patterns to support innovation, scaling, resiliency, and ecosystems
Continuously delivering high quality, more secure code to production faster and more frequently
Control and govern cost, performance and security across clusters and clouds

Figure 6 - Three Transformations Powering App Modernization

The microservices and application pattern transformation is the first. Applications must follow modern application patterns, such as microservices architectures and API-first design, to support more frequent changes, security best practices, and novel data uses (VMware Tanzu, 2021). Many organizations, however, are struggling with application modernization due to the complexity of modernizing their large application portfolios. The obvious first step in enterprise application modernization is to determine which category each of their applications falls into.

The second transformation is the DevSecOps transformation. It is all about getting the application quickly and securely into production in a continuous manner. In other terms, it refers to the requirement to accommodate a change in operating models (VMware Tanzu, 2021). DevOps approaches and continuous delivery enable companies to release higher-quality code to production quicker and more frequently to gain a competitive edge. These methods rely heavily on automation, for which a container-centric ecosystem, known as the cloud-native ecosystem, has arisen. Nonetheless, security remains critical, but it has frequently hampered speedy production releases. Incorporating security controls across the development and deployment lifecycles connects DevOps with security goals. As a result, DevSecOps has evolved as a unified strategy for organizations, establishing new criteria inside the cloud native environment.

In the Cloud Ops transformation, VMware automates data centers and accelerates cloud migrations (VMware Tanzu, 2021). Organizations can easily access elastic capacity with minimal friction, optimize infrastructure operations, shift from capital expenditure to operational spending. And thus, it is all about operating and governing platforms without coming in the way of developers.

2.3 VMware Tanzu

VMware Tanzu offers services and capabilities for modernizing your applications and infrastructure with the following objective: to continuously deliver better software to production (VMware, VMware Tanzu Intro and Vision, 2021). The portfolio streamlines multi-cloud operations while giving developers more time to work quickly and have access to the tools they need to create the greatest applications. Teams from development and operations may collaborate in novel ways using VMware Tanzu to produce game-changing business outcomes. Modern apps can be created, launched, and managed on any cloud thanks to VMware Tanzu's capabilities and services. It has features that are flexible and align with open source, such as:

- **A programming model that is easy to understand and use:** With the most widely used Java framework in the world, one can accelerate cloud-native development.
- **An experience for developers that is automated and streamlined:** With an application platform that automates the creation, release, and operation of software, one can increase developer velocity and reliably execute all applications at cloud scale.
- **Containers that are open-source and have been verified:** Upkeep a collection of open-source container images that are ready for production and have been carefully selected for developers.
- **Automation of building containers:** Automate enterprise-scale container generation, management, and governance.
- **A data layer that is modern:** With on-demand caching, messaging, and database software for modern applications, dismantle data monoliths.
- **Kubernetes that can be run in every environment:** Implement a Kubernetes runtime that is enterprise-ready to support the deployment of modern applications across on-premises, public clouds, and edge environments.
- **Unified, global multi-cluster operation:** Manage and protect Kubernetes infrastructure and modern apps across teams and clouds from a central location.
- **Microservices networking and control:** Establish end-to-end connectivity, security, and monitoring for microservices, regardless of the runtime or cloud environment.
- **Full-stack observability:** At a large scale, keep track of and examine the wellbeing of applications and infrastructure spanning multiple clouds within the enterprise, using a single data source accessible to all teams.

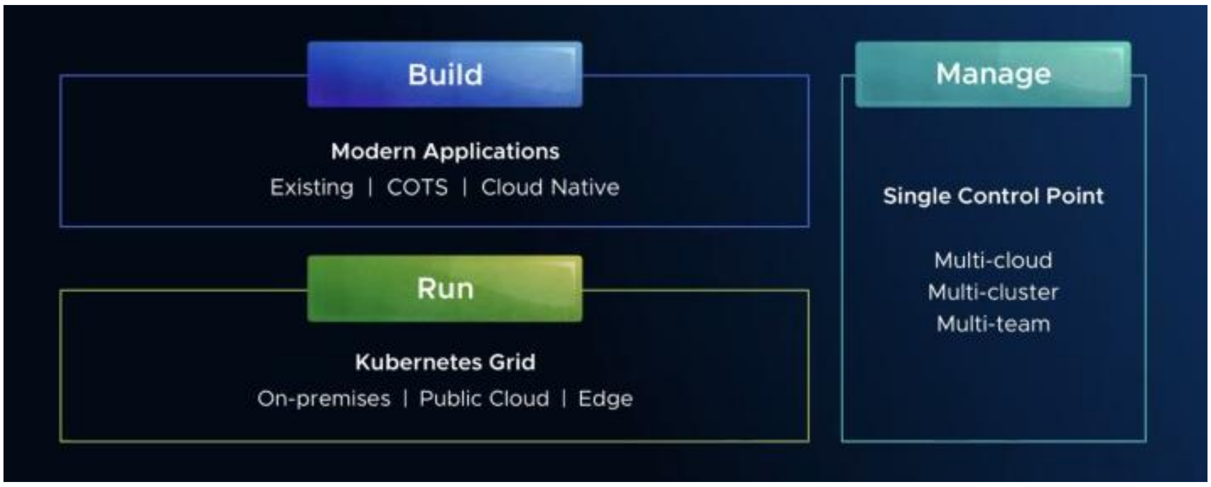


Figure 7 - VMware Tanzu Modern Applications Platform Portfolio: Build, Run and Manage Pillars

2.4 Path to Production

The following image explains the end-to-end customer journey on how to release an application. Below, the path to production will be explained while linking it to VMware products.

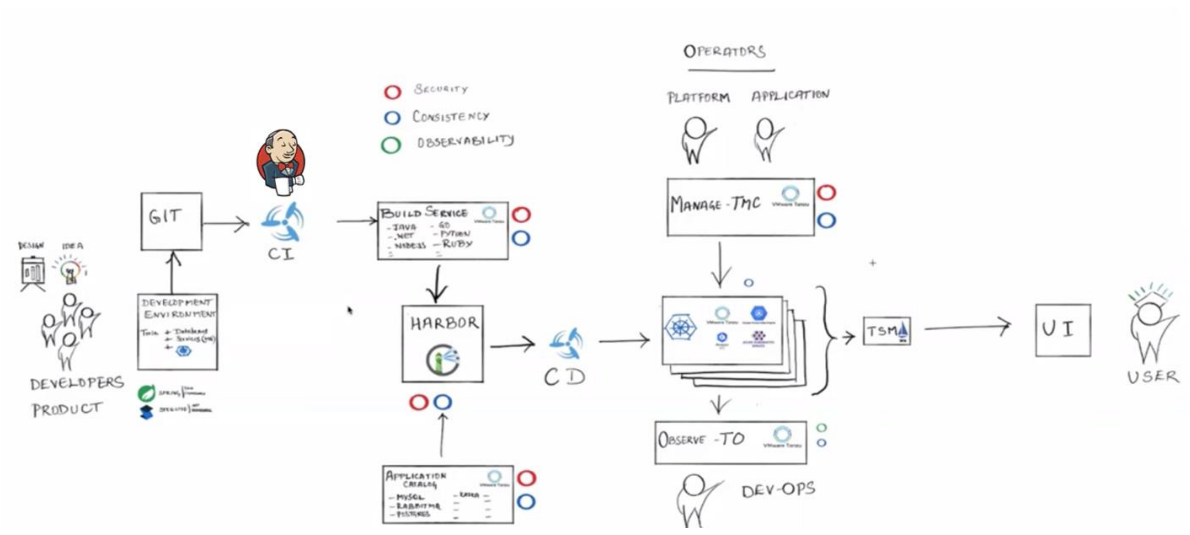


Figure 8 - VMware Tanzu: Path of an App to Container to Platform (Path to Production)

On the right-hand side of the image above, you have the end-user of the application. On the left, the business comes up with a great idea and want developers to create an application or feature. Developers are firstly going to work out what the architecture looks like, then they will start to develop code and work towards a minimum viable product. To do this, they will need a development environment, an example could be Spring, which is the number one Java development framework in the world. The code is then stored into a code repository, for example GitHub. After that, we put the code into the start of the DevOps process, the CI/CD process. The first thing we will do with the code is assess its dependency status (what libraries do we need, how do we package it all up). The Tanzu Build Service will create a container image that can then be deployed in a Kubernetes environment. Of course, the code alone is not enough to create the whole application. We also need databases, message busses, load balancers… VMware has an Application Catalog where a developer can just pull these predefined, secure, and consistent images that form microservices within your application. So once the container image is built, the developer grabs all the other components of the application that will sit in their microservices or their own containers. In other words, the application catalog is a repository of all these other elements you will need within your application, they have been curated, are security rubber stamped and directly available for use time after time. In general, the amount of open-source product that organizations use is large. We do not want developers just going out on the Internet grabbing a database and just pull it in our infrastructure, as we are unsure how the code has been affected or if it contains malware. Therefore, Tanzu's Application Catalog is becoming very popular. In the figure below, you can see the elements that you grab from Tanzu's Application Catalog, a predefined set of images with apps, caches, databases, logs, and environmental features.

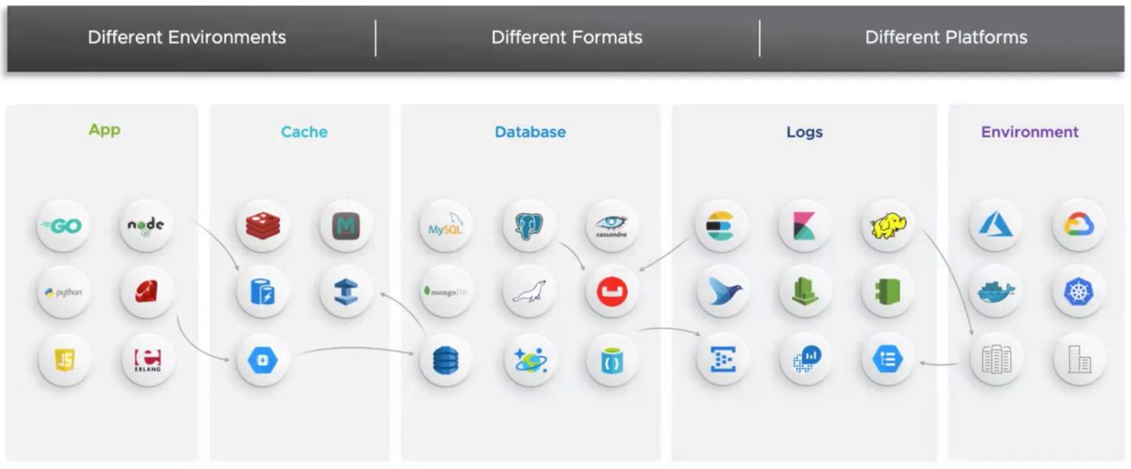


Figure 9 - Developers Build with Open-Source Images

You repeat this process for all the images that are required. We can move on to the continuous deployment (CD) phase. To deploying container images, a container engine is needed to deploy them onto, that uses a container runtime. Once the containers run, a tool to manage their lifecycle (unexpected failure, scaling etc.) is required. The industry standard is Kubernetes.

Kubernetes is an open-source platform that enables the management of containerized workloads and services in a portable and flexible manner (The Kubernetes Authors, 2021). It supports both declarative configuration and automation, making it easier to manage and scale your applications. Kubernetes boasts a rapidly expanding ecosystem with a wide range of available services, support, and tools. The name Kubernetes is derived from the Greek word for helmsman or pilot, while the abbreviation "K8s" is simply the result of counting the eight letters between the "K" and "s". Initially developed by Google, Kubernetes was open sourced in 2014. The platform combines more than fifteen years of Google's experience in managing large-scale production workloads with the best practices and ideas from the broader community.

Using containers is an effective method for packaging and running applications (The Kubernetes Authors, 2021). When running applications in a production environment, it crucial to manage the containers and minimize downtime. For instance, if a container crashes, a new one must take its place promptly. Instead of handling this manually, a system can simplify the process, and that is where Kubernetes can be helpful. Kubernetes offers a robust framework for running distributed systems, ensuring they are resilient. It automates scaling and failover for your application, deploys patterns, and performs other essential tasks. Additionally, it can easily manage a canary deployment for your system. In summary, Kubernetes provides a comprehensive solution for managing containerized applications in production environments.

- **Service discovery and load balancing**: Kubernetes offers service discovery and load balancing capabilities that allow containers to be accessed via their DNS name or unique IP address. When traffic to a container is substantial, Kubernetes can effectively distribute the network traffic through load balancing, ensuring the deployment remains stable.
- **Orchestrating storage:** With Kubernetes, one can effortlessly mount a storage system of their preference, including local storages, various public cloud providers, and more.

- **Automated deployments and reversions:** Kubernetes allow you to define the desired state of your deployed containers, and it can systematically transition the actual state to match it. This process occurs at a controlled rate, enabling you to automate tasks like creating new containers, replacing old ones, and transferring resources to new containers.
- **Automated container optimization:** By providing Kubernetes with a cluster of nodes, one can effectively manage and execute containerized tasks. Kubernetes allow you to specify the CPU and memory requirements for each container, and then automatically optimizes their placement on the nodes to maximize resource utilization.
- **Automated recovery:** Kubernetes facilitates automated recovery by restarting failed containers, replacing problematic ones, and terminating those that do not respond to user-defined health checks, and withholding them from client access until they are ready to serve.
- **Secret and configuration management:** Kubernetes enables secure storage and management of sensitive data, such as passwords, OAuth tokens, and SSH keys. With Kubernetes, you can easily deploy and update secrets and application configurations without the need to rebuild container images or expose sensitive information in your stack configuration.

Kubernetes is not a conventional, all-in-one Platform as a Service (PaaS) solution (The Kubernetes Authors, 2021). Kubernetes functions at the container level, offering a range of PaaS-like features, including deployment, scaling, and load balancing, while enabling users to integrate custom logging, monitoring, and alerting solutions. Despite its versatility, Kubernetes is not a monolithic platform, and users have the freedom to opt for alternative solutions as they see fit. By providing flexible building blocks, Kubernetes empowers developers to create personalized platforms that align with their specific needs and preferences. It does not:

- Impose no on the types of applications it can accommodate. Its primary objective is to facilitate an exceptionally broad range of workloads, encompassing stateless, stateful, and data-processing applications. Essentially, if an application is compatible with containers, Kubernetes should be able to manage it seamlessly.
- Kubernetes is not responsible for deploying source code or building applications. The design and implementation of Continuous Integration, Delivery, and Deployment (CI/CD) workflows are shaped by organizational norms, preferences, and technical needs.
- Offer application-level services, such as middleware (e.g., message buses), data-processing frameworks (e.g., Spark), databases (e.g., MySQL), caches, or cluster storage systems (e.g., Ceph) as integrated services. However, these components can be run on Kubernetes and accessed by applications running on Kubernetes through portable mechanisms like the Open Service Broker.
- Offer some integrations as examples and mechanisms for collecting and exporting metrics, but it does not enforce logging, monitoring, or alerting solutions.
- Offer or enforce a specific configuration language or system, such as Jsonnet. Instead, it provides a declarative API that can be used by any form of declarative specifications.
- Offer or adopt any complete machine configuration, maintenance, management, or self-healing systems.
- Kubernetes goes beyond just being an orchestration system and eliminates the need for explicit orchestration. Orchestration, by definition, involves executing a defined workflow where you perform A first, then B, then C. In contrast, Kubernetes consists of a collection of

independent and composable control processes that consistently push the current state towards the desired state. It does not matter how you reach from A to C, and there is no need for centralized control. This approach results in a more flexible, powerful, resilient, and scalable system that is easier to use.

Tanzu Kubernetes Grid (TKG) is the Kubernetes distribution VMware sells and is part of the Tanzu portfolio. It would also be possible to use AKS, RedHat OpenShift or eventually Docker Swarm (non-Kubernetes). Tanzu Kubernetes Grid is designed with the aim of streamlining the installation process and simplifying Day 2 operations. It integrates essential open-source technologies and automation tools to enable you to quickly establish a scalable, multi-cluster Kubernetes environment. Therefore, it can be considered as a platform to build platforms. It is tightly integrated with vSphere and can be extended to run with consistency across your public cloud and edge environments. However, TKG deployments are managed using command line interface (CLI) tool and is thus can be quite tricky to manage. An easier interface is needed, and this is where Tanzu Mission Control (TMC) comes in. VMware Tanzu Mission Control is a platform that offers centralized management for Kubernetes infrastructure and modern applications, ensuring consistent operation and security across multiple teams and cloud environments. By providing a single control point, operators can manage numerous clusters, including different distributions beyond TKG. Meanwhile, developers can leverage a self-service platform to drive business forward while maintaining consistent management and operations across environments, leading to enhanced security and governance.

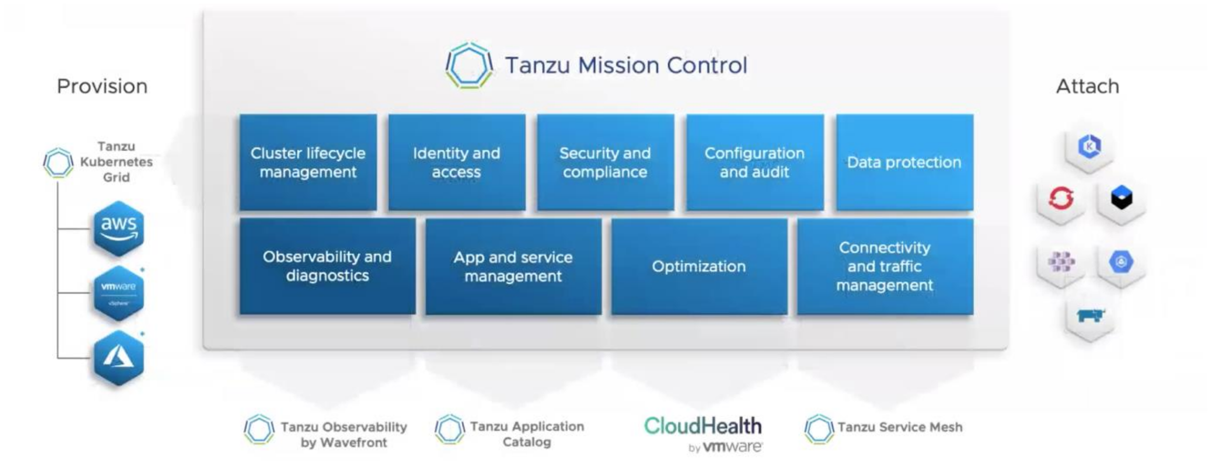


Figure 10 - Tanzu Mission Control: A Centralized Kubernetes Management Platform Across Clouds

Lastly, containers are discrete entities. Unless you tell them about each other, they do not know about each other. We need to mesh these containers together so they know who they can talk to, what protocols they are allowed to use, what data flows between them and in what direction. This is where Tanzu Service Mesh comes in. The Service Mesh capability allows you to connect effectively all these containers and clusters together so that each microservice can talk to each other, has the right access rights and is secure while keeping to simple for the developer. In essence, VMware Tanzu Service Mesh delivers consistent security and operations throughout the entire application transaction process, from end-users to services to data, across any platform or cloud. Tanzu Service Mesh provides consistent connectivity and security for microservices – across all your Kubernetes clusters and clouds. Then, we want to observe the whole stack in real-time with Tanzu Observability. VMware Tanzu Observability delivers monitoring, observability, and analytics for cloud-native applications and environments.

DevOps, site reliability engineers and developer teams use Tanzu Observability to proactively alert on, rapidly troubleshoot and optimize performance of their modern applications running across vSphere and multi-cloud environments. This allows us to get insights in the performance of the application, where the issues are and what we need to change. When we worked out what needs optimizing, we want to work around this cycle again as fast as possible. This process used to take a year with a monolithic architecture, now we can go through this process with customers in hours. With the whole portfolio, we drive security, consistency, and full visibility in real-time across the whole application release and the life cycle management of it.

With such a context in mind, the objective of this practical experience is to seek internship-specific contributions to the above-mentioned digital transformations. As a digital solution engineer, one aims to help customers conquer the challenges of the future through finding new ways to deliver value so they can succeed in those changing economic conditions. In other words, it is to help them become more efficient and future-proof through the adoption of a variety of technology solutions to be one step ahead of their competitors.

3. METHODOLOGY, TECHNOLOGIES, AND TOOLS

3.1 METHODOLOGY

To develop the application, I have split the work in different phases, namely a training phase, a handover phase, a research phase, a design phase where I split the application in different microservices, the development phase, and lastly the deployment and testing phase. In this section, I will briefly explain what I did during these different phases.

VMware has vastly increased its investments in training solution engineers. Before I started any work, I enjoyed trainings that introduced VMware and its technologies. Afterwards, I had an in-role training as part of the solution engineering team. I still needed to improve my technical knowledge on the job and thus it was necessary to conduct more theoretical research on what a cloud native application is, and on the different technologies involved.

After my training period, there was an official handover phase with the previous trainee. This phase was crucial since it was the previous trainee who entirely developed the heart of the application. He developed the neural network allowing to analyze the images uploaded by the user. The trainee told me everything about the architecture of the application and the difficulties he encountered.

Initially, the objectives for improving the application were the following. First, I updated the application architecture so that it is in line with the fundamental principles of a distributed microservices application, with the twelve factors. I switched from completely synchronous processing to synchronous reading (leveraging APIs) and asynchronous writing through a message queue and message broker. I needed to make the application deployable on TKG which is VMware's Kubernetes solution. The application needed to run fully in Docker containers. Lastly, it was my responsibility to fully document the deployment and functioning of the application to facilitate the task of the future employee who will have to further improve it by connecting it with external applications through APIs.

To achieve the above requirements, I divided the work in different phases that were logic to build the application. After my training and during my handover phase, I officially started the research phase. This phase overlapped with the previous one as I quickly realized that almost all the subjects I would have to deal with and the technologies I would use were new to me. I read a lot of technical documentation, attended conferences, took theoretical courses to learn about different programming languages. The combination of different resources made it easier for me to understand relatively complex notions. The most effective way for me to learn was through combining these resources, then I executed some tests and finally, if necessary, I consulted the technical documentation. After the documentation stage, the design, development, and deployment and testing phase followed. In the design phase, I made the global architecture and composed the actual breakdown of the different microservices. The first microservice is the services analysis, which had to be written in Python, as the core of the application uses the Python library PyTorch. Then I separated the read and write queries, therefore we have a separate microservice for the claim queries and claim commands (writes). The service for claim handling is a separate one that stores the command and processes it. This creates claims in the database and is the only service that writes the database. In the development phase, we can make the distinction between front and back-end. The main tools used for the front-end development are Angular with VMware's Project Clarity (UI/UX) module. The back end is written in

Python and NodeJS. These technologies and tools are explained in section 3.2. In the last phase, the deployment and testing phase, I used single replicas of each microservice to check if the application works as intended. Afterwards, I increased this number to check the load balancing functionalities.

3.2 Technologies & Tools

3.2.1 The MEAN Stack

The MEAN stack is a newly emerging web development stack (Nirgudkar & Singh, 2017). The MEAN stack is comprised of four technologies: MongoDB as the database, Express as the server framework, Angular for the front-end, and Node.js as an event-driven I/O server-side JavaScript environment. What sets the MEAN stack apart is that all four technologies are based on JavaScript and JSON, which allows for efficient data exchange between them, without the need for JavaScript Object Notation (JSON) encoding. The learning curve for the MEAN stack is steep, as it only requires knowledge of JavaScript, rather than multiple programming languages. One of the benefits of using the MEAN stack is Node's package ecosystem, NPM, which boasts the largest collection of open-source libraries. Node uses JavaScript as its programming language for both server and client-side. MongoDB serves as the database to store the application's necessary data, Angular is the client-side front-end application, Express is a lightweight HTTP server framework that serves the Angular application and its resources, and Node.js is the environment that runs Express. Each of these technologies will be explained in greater detail below.

3.2.1.1 MongoDB

In developing the application, I utilized MongoDB, a general-purpose database that offers significant power, flexibility, and scalability (Bradshaw, Brazil, & Chodorow, 2019). MongoDB boasts a unique combination of scaling capabilities and advanced features such as secondary indexes, range queries, sorting, aggregations, and geospatial indexes. It differs from relational databases in that it is document-oriented, which enables greater flexibility when scaling out. This approach replaces the traditional row-based model with a more versatile document-based model that allows for embedded documents and arrays. With no predefined schemas, adding or removing fields becomes a hassle-free task, which accelerates development cycles and allows for experimentation with different data models. As data set sizes continue to expand at an exponential pace, developers need to determine how best to scale their databases. While scaling up with larger machines is one option, it eventually becomes prohibitively expensive. Scaling out, on the other hand, is more scalable and affordable, but it also adds complexity to administration. MongoDB was built with scaling out in mind and uses its document-oriented model to split data across multiple servers. It automatically balances data and load across a cluster and routes user requests to the appropriate machines, reducing administrative overhead. MongoDB's performance is also exceptional, thanks to dynamic padding, preallocation of data files, and efficient use of cache memory.

3.2.1.2 Express.js

Express.js is a web framework that utilizes the fundamental Node.js http module and connect components (Mardanov, 2014). The framework's philosophy is centered around configuration over convention, with its cornerstone being the middlewares - components that developers can select to suit the requirements of their project. This approach provides developers with the flexibility and customization they need. If developers rely only on the core Node.js modules, they may find themselves constantly reinventing the wheel, writing the same code for similar tasks such as parsing HTTP request bodies, and organizing routes based on URL paths and HTTP methods using if conditions. Express.js resolves this issue and offers a model-view-controller (MVC) architecture for web applications. These applications can range from simple back-end REST APIs to complex, scalable, and real-time full-stack web apps.

Express.js usually has an entry point, a.k.a., a main file (Mardanov, 2014). In that file, the following steps are performed:

- Incorporate external dependencies along with proprietary modules, such as models, helpers, controllers, and utilities.
- Set up the app configurations in Express.js, which includes specifying the file extensions for the template engine and its associated files.
- Specify various middleware, such as error handlers, parsers for cookies and other data, and the folder for serving static files.
- Define routes.
- Connect to databases, in this case MongoDB.
- Start the app.

As the Express.js application runs, it actively receives and processes incoming requests (Mardanov, 2014). As requests arrive, they are sequentially processed through a predetermined chain of middleware and routes, starting from the top and progressing downwards. This feature is critical because it enables developers to manage the execution flow.

3.2.1.3 Angular

The main language used for the front-end of my web application Tito is Angular. AngularJS belongs to a new wave of libraries and frameworks aimed at facilitating the development of web applications that are more efficient, adaptable, maintainable, and testable (Branas, 2014). AngularJS is a client-side JavaScript framework that was founded in 2009 by Miško Hevery and Adam Abrons. It is an open-source framework that aims to enhance web development productivity. AngularJS adopts a declarative approach to build user interfaces, while imperative programming is used to implement an application's business logic. The framework extends HTML's vocabulary, making development easier for programmers, resulting in more maintainable, reusable, and expressive application components, reducing the amount of unnecessary code, and keeping the team focused on valuable tasks. To set up AngularJS, the configuration process is straightforward. The angular.js script must be included in the HTML templates, and the application module can be created by calling the module function with its name and dependencies from Angular's API.

3.2.1.4 Node.js

Node.js is a back-end JavaScript runtime environment that operates on the V8 engine, is open-source and cross-platform, and runs JavaScript code without requiring a web browser (Wikipedia, Node.js, 2021). With Node.js, developers can utilize JavaScript for creating command line tools, as well as server-side scripting, where scripts are executed on the server-side to generate dynamic web page content before it is delivered to the user's web browser. This creates a JavaScript everywhere approach, bringing together web-application development under one programming language rather than using different languages for client-side and server-side scripts.

3.2.2 Apache Kafka

Companies that have an online presence and engage in web-based activities generate a significant volume of data (Garg, 2013). Data is a relatively new component of internet-based systems, and it is typically managed through logging and traditional log aggregation solutions due to the high amount of data being processed. These solutions are effective for offline analysis systems like Hadoop, but they are limiting for real-time processing. The large volume of data being collected and processed from production systems makes it difficult to use this data in real-time. Apache Kafka aims to address this challenge by enabling parallel loading in Hadoop systems and partitioning real-time consumption across a cluster of machines. Although Kafka is useful for processing activity stream data like Scribe or Flume, its architecture is closer to traditional messaging systems like ActiveMQ or RabbitMQ. Applications generating real-time information are often far removed from those consuming it, which can result in the need for redevelopment or integration between the two. A seamless integration mechanism is needed to avoid rewriting applications at either end. In the current era of big data, the foremost challenge is collecting the data due to its enormity, followed by analysing it. For the communication between the microservices, I used Apache Kafka as a message broker. The act of message publishing involves utilizing messages to establish a connection between different applications, with the aid of a message broker such as Kafka that routes the messages between them (Garg, 2013). Kafka offers a real-time solution for software systems to handle large volumes of information and distribute it quickly to multiple consumers without blocking the producers or revealing the identity of the final consumers. It is an open-source messaging system that is designed with several key features. Firstly, Kafka uses O(1) disk structures that enable constant-time performance even when dealing with vast amounts of data. This ensures that data is persistent, and no information is lost. Secondly, Kafka is designed to function on commodity hardware and handle millions of messages per second, resulting in high throughput. Thirdly, Kafka supports message partitioning and distribution over a cluster of consumer machines while maintaining per-partition ordering semantics, making it a distributed system. Additionally, Kafka offers multiple client support, allowing for seamless integration with clients from different platforms. Finally, Kafka allows for immediate visibility of messages produced by producer threads to consumer threads, making it an ideal solution for event-based systems. In summary, Kafka provides a real-time publish-subscribe solution that effectively addresses the challenges of using real-time data consumption for large data volumes.

3.2.3 Project Clarity

Project Clarity is a design system that integrates UX guidelines, an HTML/CSS framework, and Angular components, and is available as an open-source solution (Project Clarity, sd). Both designers and developers can benefit from Clarity. Clarity's designs are based on ongoing exploration and research, and they are incorporated into HTML/CSS components that can be used by any web UI, regardless of the underlying JavaScript framework. Additionally, Clarity provides a collection of well-designed and implemented data-bound components that are built on top of Angular, which is one of the most popular JavaScript frameworks available. I utilized Clarity to expedite front-end development for this project.

4. DEVELOPED ACTIVITIES

4.1 Different phases of work

Once I had a clearer idea on the different technologies and solutions I was going to adopt for my application, I started the design phase of the general architecture. This section focuses in more detail on how the developed application is organized. To better understand the architecture of the application, the literature review focuses on the essence of a distributed cloud native application. This foundation was laid by the so-called twelve-factors, a list of service characteristics that were defined in relation to the needs of a distributed application, intended to be deployed in the cloud. The most relevant criteria for the development of my application are the following:

- Single source code for multiple deployments.
- Explicit declaration of program dependencies.
- Program configuration provided by environment variables.
- Services must be stateless programs. Identical output for identical input.
- Easy and fast stopping and starting of services.
- Scaling service by service.

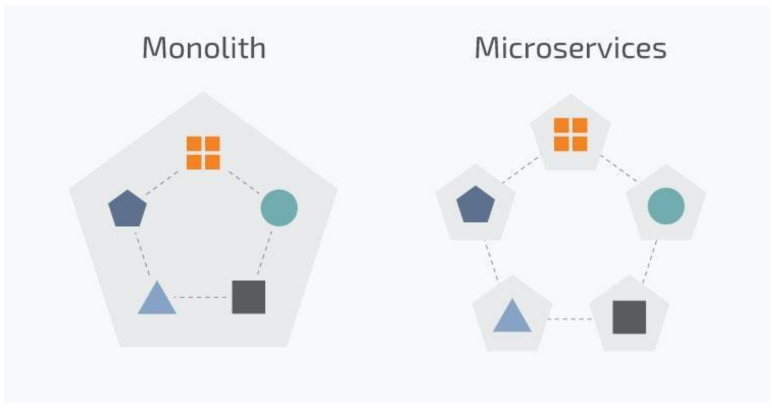


Figure 11 - Monolithic Architecture vs. Microservices

As the architecture and functioning of a distributed cloud native application is now understood, we can dive deeper into the global architecture of my application. Since my application is a microservices application, I had to fragment the global logic of the application into different pieces performing one or more tasks related to the same functionalities. As each piece is a containerized microservice, I was able to choose the most adapted language each time (e.g., to query and update the database, Express.Js and MongoDB are used, which are JavaScript frameworks/libraries and thus the language used is Node.JS. As these different services are connected on the same network, it was necessary to choose the communication protocols and compression (e.g., HTTP API requests compressed in gzip). During this phase it was also necessary to consider all the difficulties that a distributed architecture can cause, i.e., unavailable services, mostly stateless.

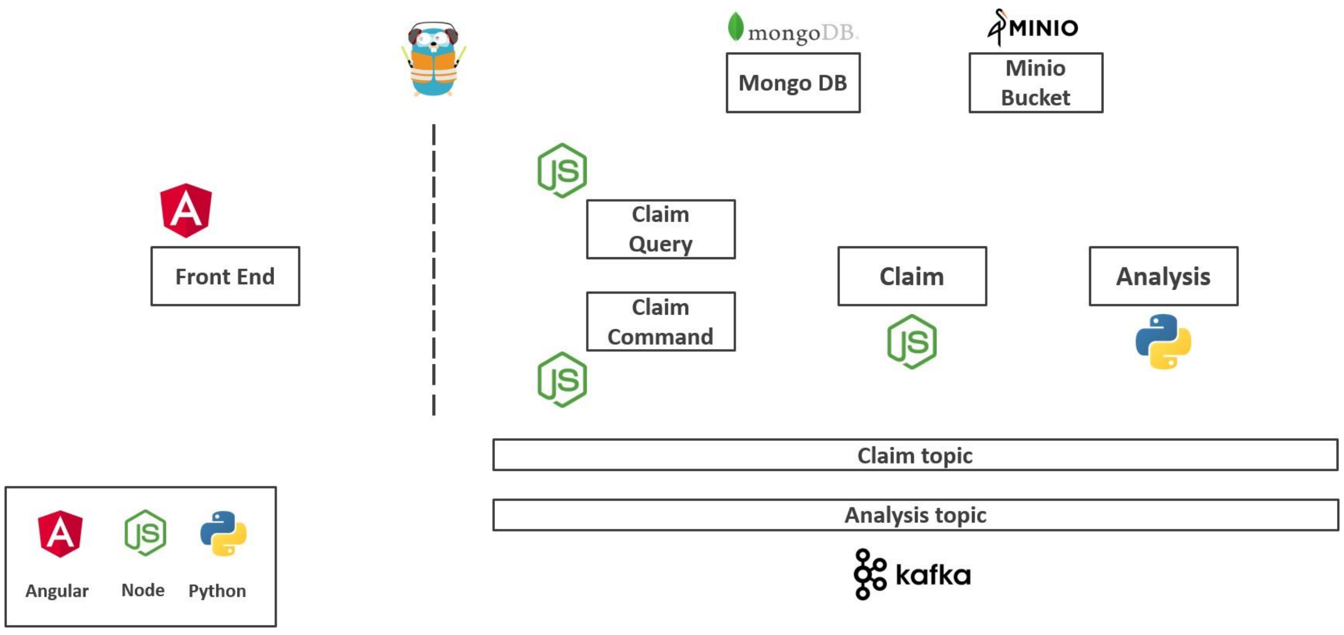
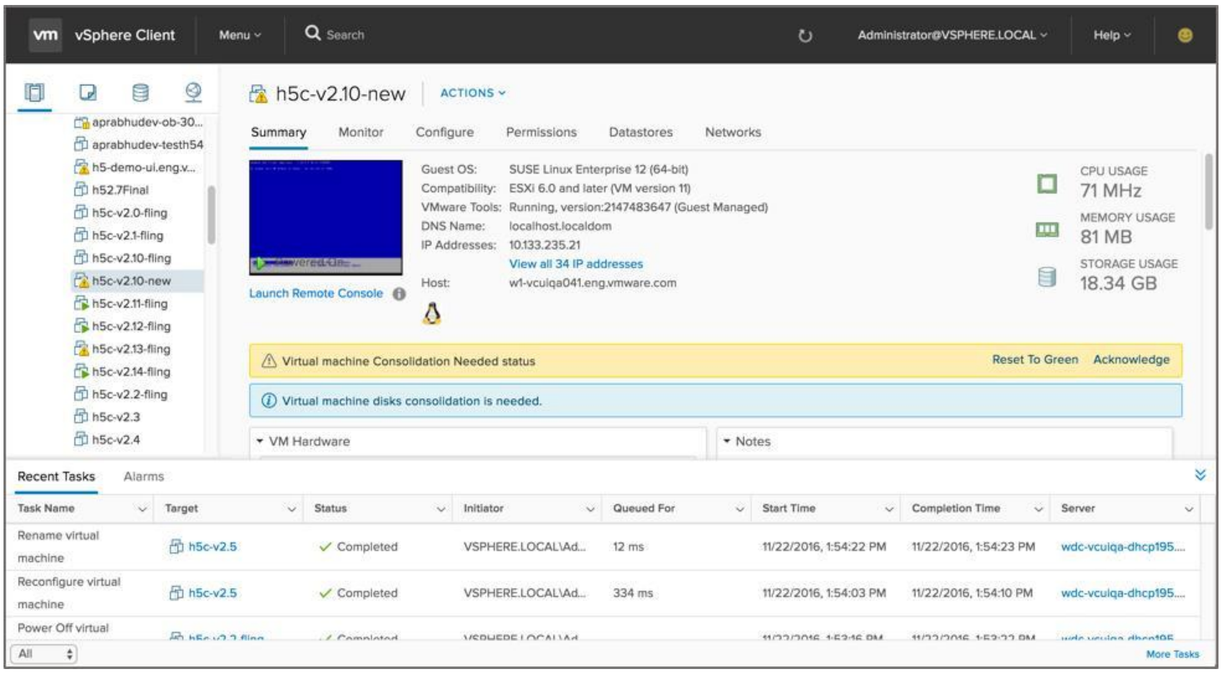


Figure 12 - Global Architecture of the Application

As mentioned previously, there was already an existing demo I would start with. The latter, called Tito, allowed to calculate the travel time of its users and in a new version, a functionality to reserve rental cars to make these same trips was added. My objective was to create an application that could be added to the existing one by including an automation process for the return of the rented vehicles. When handing in a rental car, it needs to be analysed for damages realised by the lessee. In the app, it would be a matter of sending a picture of the vehicle which would then be analysed to determine if it was damaged or not.

For the software part, I started with the MEAN (MongoDB, Express.js, Angular, Node.js) Stack. First, MongoDB was particularly relevant in this scenario as it is a NoSQL database extremely resistant to load and functioning in a distributed way either thanks to the principle of sharding which allows us to linearly scale thanks to the distribution of the documents of a collection on several nodes or of replication. It is the last that was adopted. Second, Express.js is a Node.js web framework allowing to parse and process HTTP requests. It is thanks to Express.js that services can expose APIs to communicate between them. Angular is a JavaScript framework used for front-end web development. It is modular thanks to its system of components, modules, and services. HTML templates are created that will allow the display of information on a page according to a logic defined in a Typescript program. It is possible to directly call the variables of the program in the templates and iterate HTML tags on Typescript lists for example. Lastly, Node.js is a very widespread back-end web framework since it is powerful. It is lightweight and takes advantage of multithreading by running sometimes asynchronously. This allows in the case of web requests to process more than one at a time and to better manage latency and data transfer time on the network (e.g., processing a photo upload as in our application which is triggered only after receiving it in its entirety). For the front-end, in addition to the Angular framework I used a second overlay, the Clarity framework by VMware. VMware Project Clarity is a project aiming to simplify the development of graphical interfaces to improve the user experience (UX) without much overhead for the developer. This framework is used today for a multitude of VMware products and allows users not to feel confused when they switch from one interface to another. The objective of Project Clarity is to rely on the intuitive character of the interface. Project Clarity is offered as an Angular module. The developer can then reference Clarity objects directly in his templates (e.g., <clr-form> for a form).



Guest OS: SUSE Linux Enterprise 12 (64-bit)
Compatibility: ESXi 6.0 and later (VM version 11)
VMware Tools: Running, version:2147483647 (Guest Managed)
DNS Name: localhost.localdom
IP Addresses: 10.133.235.21
View all 34 IP addresses
Host: w1-vcuiqa041.eng.vmware.com

CPU USAGE 71 MHz
MEMORY USAGE 81 MB
STORAGE USAGE 18.34 GB

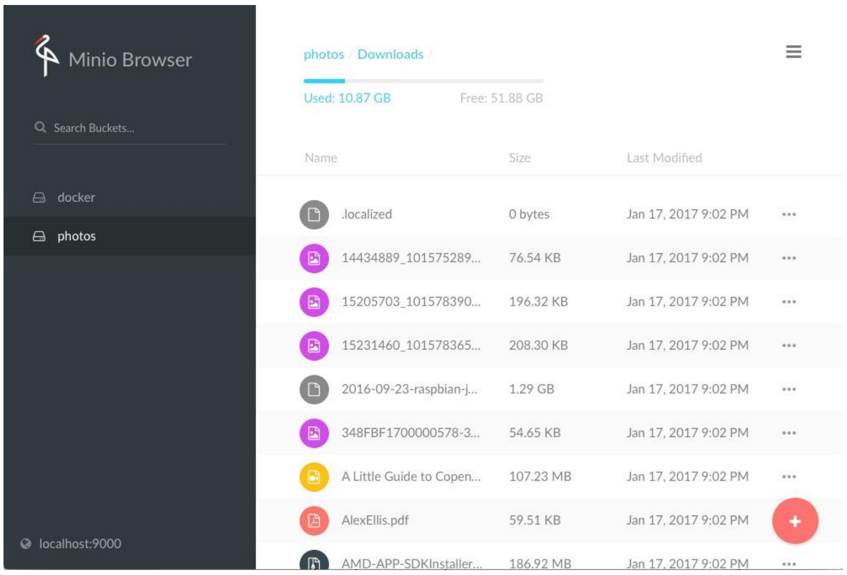
Virtual machine Consolidation Needed status — Reset To Green — Acknowledge

Virtual machine disks consolidation is needed.

Task Name	Target	Status	Initiator	Queued For	Start Time	Completion Time	Server
Rename virtual machine	h5c-v2.5	✓ Completed	VSPHERE.LOCAL\Ad...	12 ms	11/22/2016, 1:54:22 PM	11/22/2016, 1:54:23 PM	wdc-vcuiqa-dhcp195....
Reconfigure virtual machine	h5c-v2.5	✓ Completed	VSPHERE.LOCAL\Ad...	334 ms	11/22/2016, 1:54:03 PM	11/22/2016, 1:54:10 PM	wdc-vcuiqa-dhcp195....
Power Off virtual	[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]

Figure 13 - Interface of VMware vSphere: Implementing Project Clarity

HTTP requests are useful when an instant response is needed, for example if the database wants to read something. However, while waiting for the data to be read from the database, the user is still waiting. If I want to request the analysis of the car I just returned, I do not necessarily have to wait for this request to be processed. I just need to know that my photo is saved and that my request has been registered. To obtain this result, Minio and Apacha Kafka are used. First, Minio stores images waiting to be processed. Minio is an AWS S3 compatible storage. This storage is accessible via the internet and protected by authentication keys. The advantage of Minio storage is that it can download and upload files from many languages thanks to officially supported clients and other clients developed by the community. It uses the S3 API in the same way as the AWS S3 bucket and offers the possibility to be hosted on a multitude of different media such as the private cloud, public cloud and Docker. It is possible to deploy Minio on Kubernetes in a distributed way.



Name	Size	Last Modified
.localized	0 bytes	Jan 17, 2017 9:02 PM
14434889_101575289...	76.54 KB	Jan 17, 2017 9:02 PM
15205703_101578390...	196.32 KB	Jan 17, 2017 9:02 PM
15231460_101578365...	208.30 KB	Jan 17, 2017 9:02 PM
2016-09-23-raspbian-j...	1.29 GB	Jan 17, 2017 9:02 PM
348FBF1700000578-3...	54.65 KB	Jan 17, 2017 9:02 PM
A Little Guide to Copen...	107.23 MB	Jan 17, 2017 9:02 PM
AlexEllis.pdf	59.51 KB	Jan 17, 2017 9:02 PM
AMD-APP-SDKInstaller...	186.92 MB	Jan 17, 2017 9:02 PM

Figure 14 - Web Interface of Minio Storage

To manage communications between services that are asynchronous, a message broker, Apache Kafka, is used. Apache Kafka is also deployed in a distributed way. Kafka allows to exchange messages on topics. A program can become a producer and/or a consumer of messages for one or more topics.

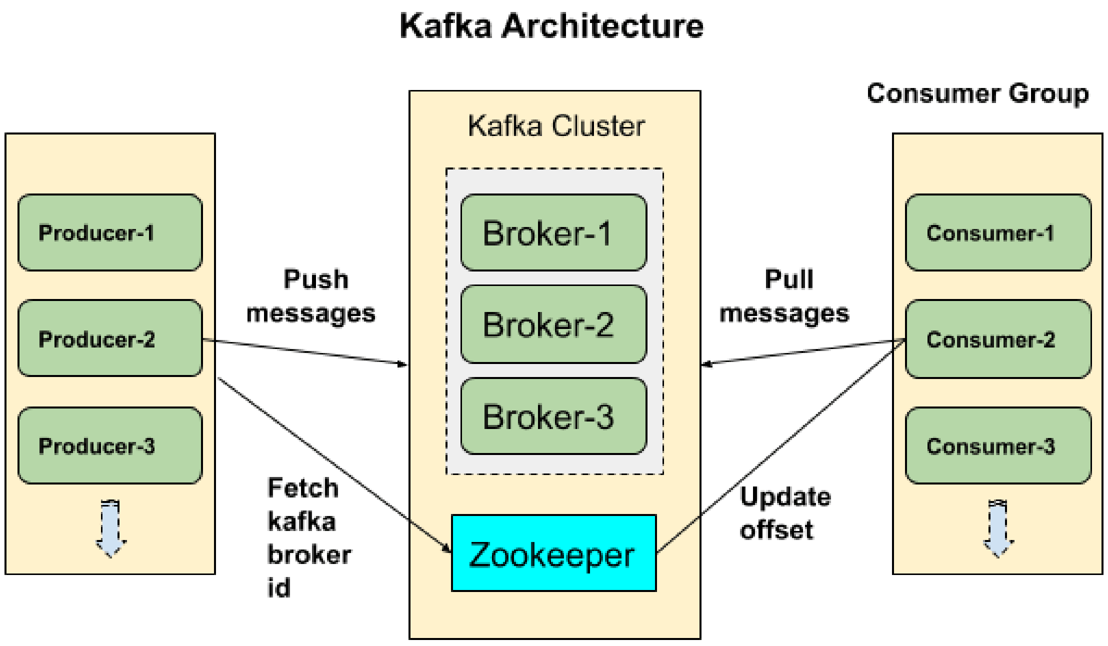


Figure 15 - Apache Kafka Architecture

When a message is stored in Kafka, it will be consumed "at least once"; it is replicated on the different nodes of the cluster. It is possible, thanks to settings, to prioritize, for each subject, the availability of the cluster or the consistency of the data. I have personally opted for availability. It is possible to store and then compact all the messages thanks to the creation of a topic. This allows to move back to the previous state of the system in case of a database corruption. It can also allow to perform later analysis of the messages for example for an audit (Extract, Transfer, Load - ETL). Indeed, the CAP theorem stated by Eric Brewer tells us that it is not possible for a distributed system to be both consistent, available, and resistant to a partitioning of a part of the network. We must choose at most two properties among the three.

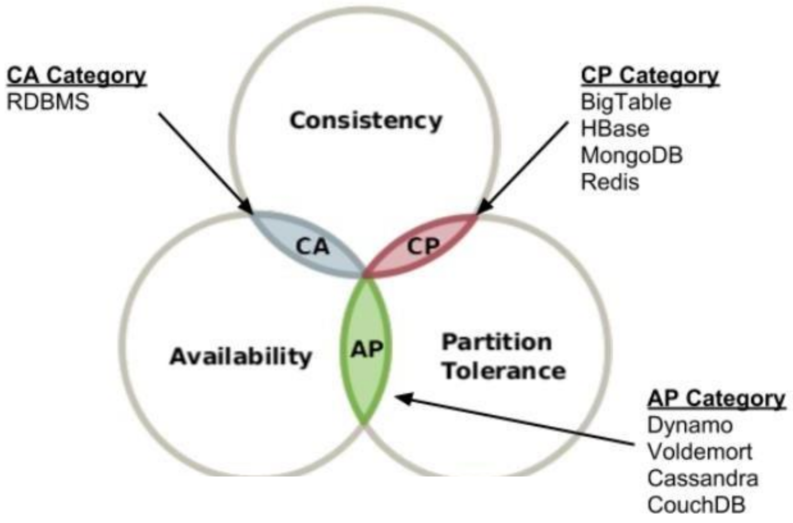
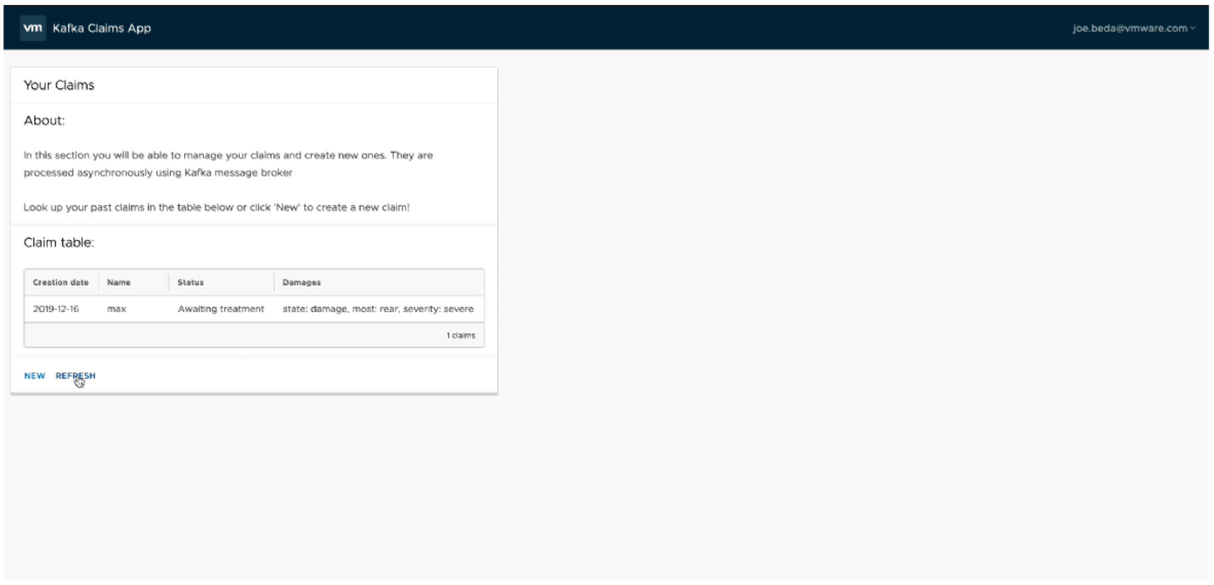


Figure 16 - CAP Theorem

Kafka, Minio, MongoDB and Traefik are deployed on the Kubernetes cluster using Helm. Helm is a tool for creating reusable templates or charts of Kubernetes deployments. It allows to speed up this process while keeping the possibility to set up what we want. The charts used come from the default repository but also from Tanzu Application Catalog, a catalog of ready-to-use Kubernetes applications packaged as Helm charts. This is a great tool to make the use of Kubernetes more accessible in a professional environment. We can use Helm to quickly deploy everything we need on a freshly created environment as well (EKS type).

To better illustrate the two-phase operation of the application, one synchronous and the other asynchronous, I will describe the different stages of processing a write request (new request):

We start by sending the vehicle photo submission form from the web interface (Angular). The Claim Query service receives the request and uploads the photo to the Minio bucket and then sends a Kafka message to the Claim Service with the format {type: " new ", claim_id: id, content: contentOfForm} on the Claim Topic. Once the message is received by Kafka, the client is informed and **the synchronous processing phase ends**.



Your Claims

About:

In this section you will be able to manage your claims and create new ones. They are processed asynchronously using Kafka message broker

Look up your past claims in the table below or click 'New' to create a new claim!

Claim table:

Creation date	Name	Status	Damages
2019-12-16	max	Awaiting treatment	state: damage, most: rear, severity: severe

1 claims

Figure 17 - Clarity Interface of the Application

The asynchronous processing phase (Event Driven) begins. When the Claim Service consumes the "new" message, it will create an entry in the MongoDB database for this request and then produce a Kafka message on the Analysis Topic to request the image analysis. The Analysis Service will then consume this message and download the corresponding image from the Minio bucket. Once the image is analysed, the service will produce an "update" message on the Claim Topic. This service is completely developed in Python. Indeed, the image analysis is made possible by the PyTorch library, which is very well known.

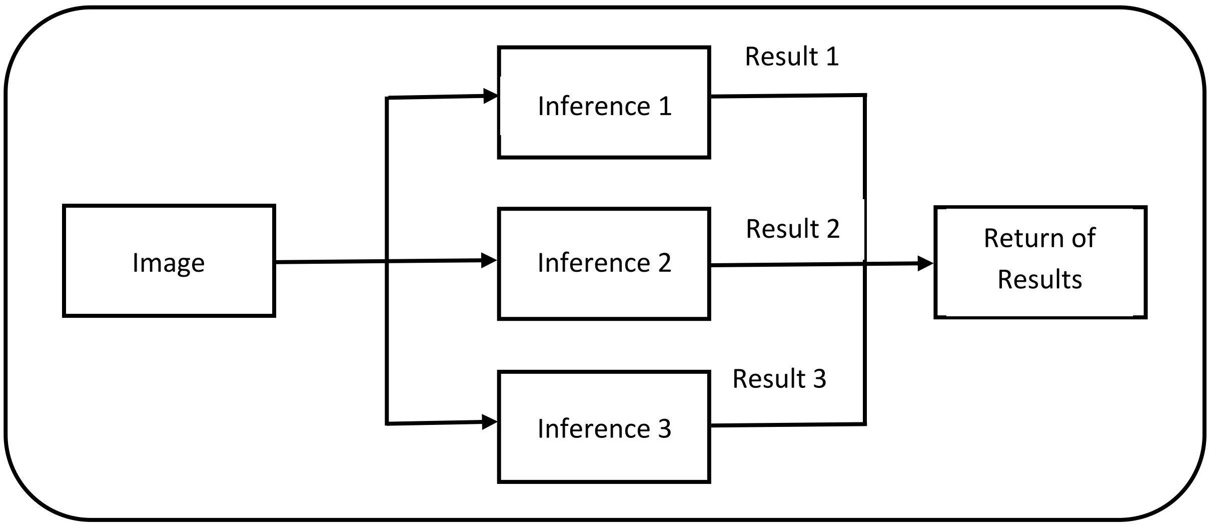
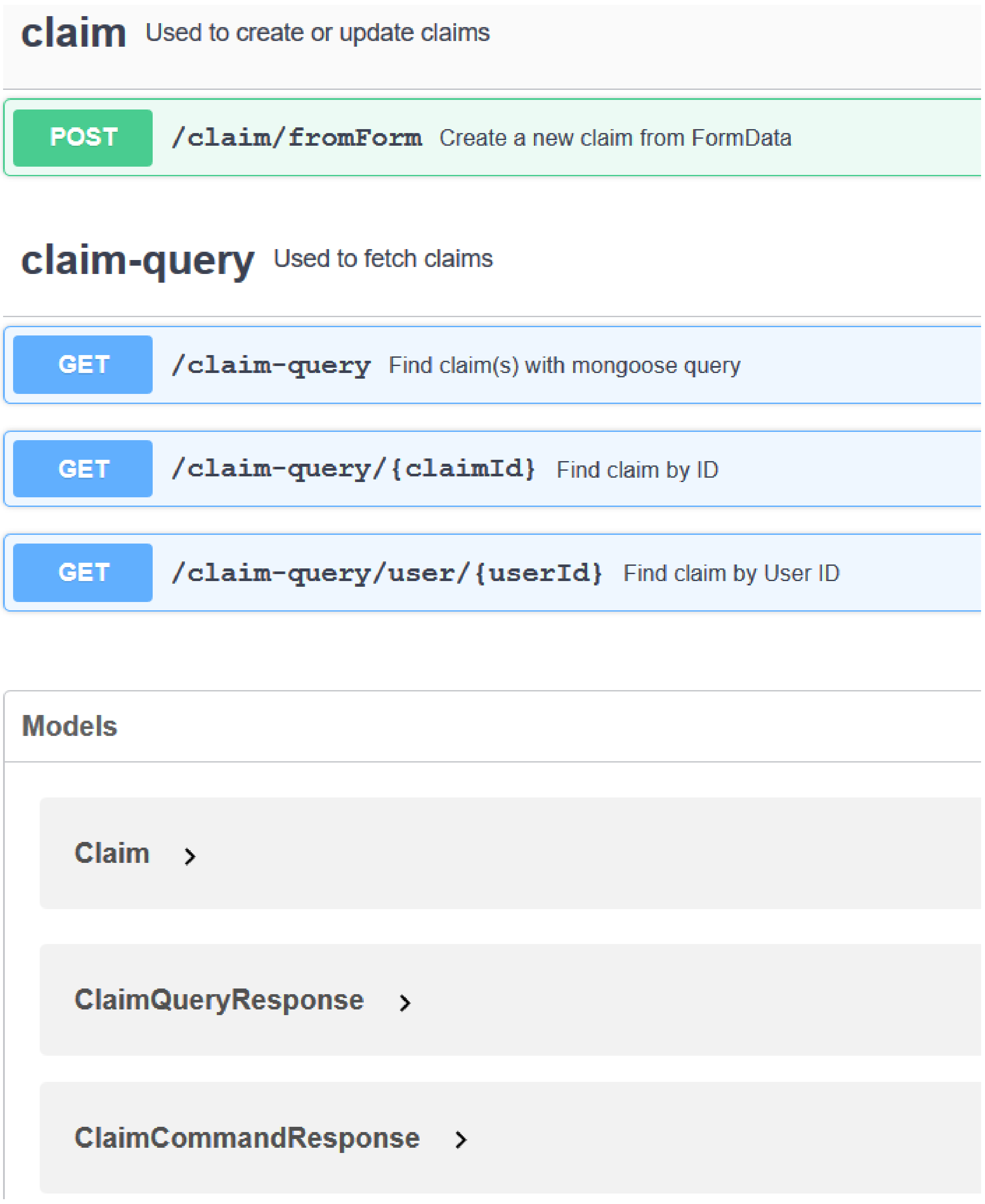


Figure 18 - Logic of the Analysis Service Script

When this same message is consumed by the Claim Service the MongoDB entry will be updated with the results of the analysis. It will be very easy to integrate this system in the next version of Tito simply by removing the Angular microservice and connecting to the API from the Tito interface. The documentation of this API can be consulted thanks to Swagger (description of routes, parameters, formats of the answers etc.)


claim Used to create or update claims

POST /claim/fromForm Create a new claim from FormData

claim-query Used to fetch claims

GET /claim-query Find claim(s) with mongoose query

GET /claim-query/{claimId} Find claim by ID

GET /claim-query/user/{userId} Find claim by User ID

Models

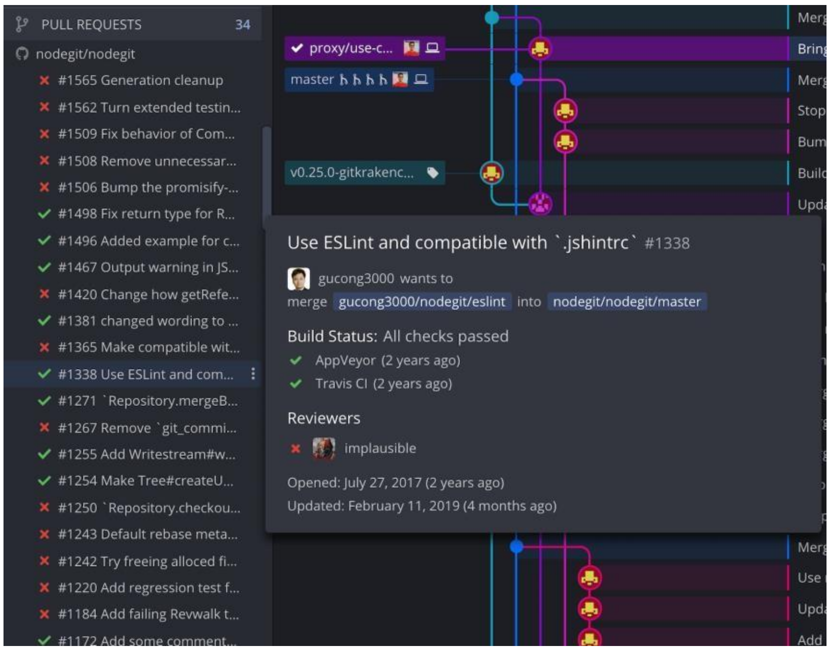
Claim >

ClaimQueryResponse >

ClaimCommandResponse >

Figure 19 - Swagger UI

After the design, it was necessary to start developing the application. For the development, I chose to develop each microservice separately and sequentially meaning I did not move to the next until one was finished. Since I had to use different programming languages and different libraries, I needed to go back to theoretical or syntactic notions more than once. Also, to manage the development as well as possible and to limit the risks of disaster (loss of the work done for example), a git with online backup (Gitlab) has been created. To facilitate the use of git, in addition to the management via the command prompt I took advantage of a utility proposing an interface of visualization of git: GitKraken. This tool was advised to me by the trainee I replaced.


PULL REQUESTS 34
nodegit/nodegit
- ✗ #1565 Generation cleanup
- ✗ #1562 Turn extended testin...
- ✗ #1509 Fix behavior of Com...
- ✗ #1508 Remove unnecessar...
- ✗ #1506 Bump the promisify-...
- ✓ #1498 Fix return type for R...
- ✓ #1496 Added example for c...
- ✓ #1467 Output warning in JS...
- ✗ #1420 Change how getRefe...
- ✓ #1381 changed wording to ...
- ✗ #1365 Make compatible wit...
- ✓ #1338 Use ESLint and com...
- ✓ #1271 `Repository.mergeB...
- ✗ #1267 Remove `git_commi...
- ✓ #1255 Add Writestream#w...
- ✓ #1254 Make Tree#createU...
- ✗ #1250 `Repository.checkou...
- ✗ #1243 Default rebase meta...
- ✗ #1242 Try freeing alloced fi...
- ✗ #1220 Add regression test f...
- ✗ #1184 Add failing Revwalk t...
- ✓ #1172 Add some comment...

Use ESLint and compatible with `.jshintrc` #1338

gucong3000 wants to merge gucong3000/nodegit/eslint into nodegit/nodegit/master

Build Status: All checks passed
- ✓ AppVeyor (2 years ago)
- ✓ Travis CI (2 years ago)

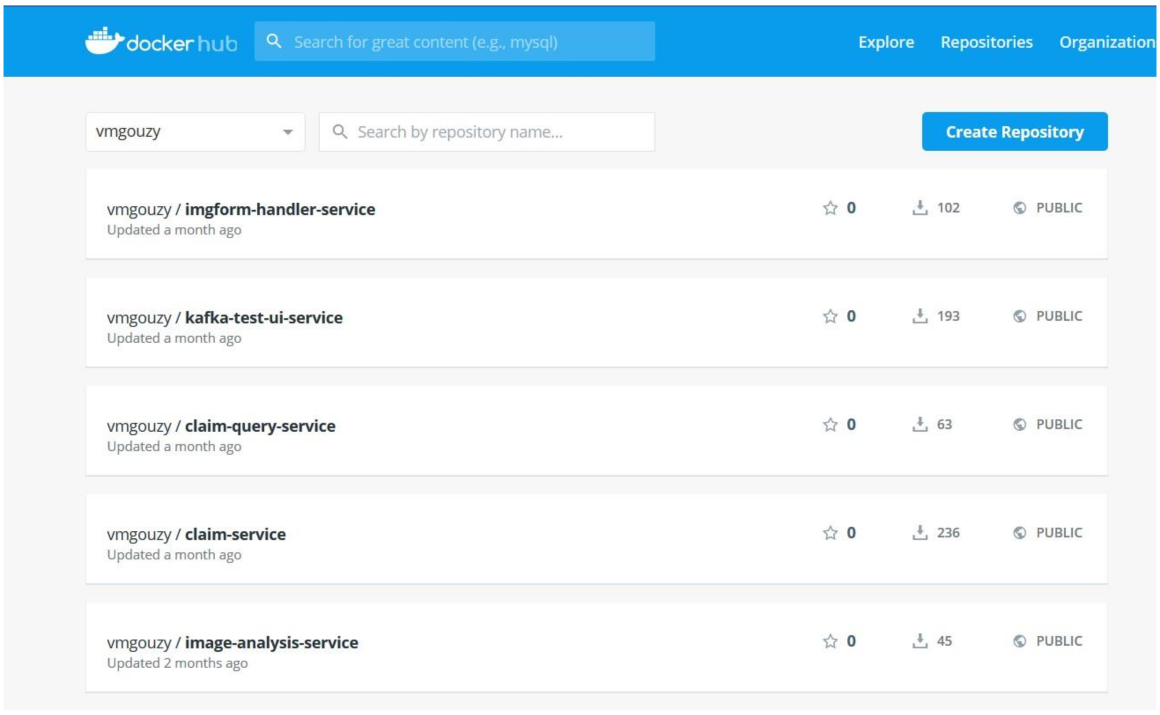
Reviewers
- ✗ implausible

Opened: July 27, 2017 (2 years ago)
Updated: February 11, 2019 (4 months ago)

Figure 20 - Interface GitKraken: Pull Requests

It was necessary to have a maximum of parity between the development environment and the production one. At the beginning, the command "ng serve" of the Angular binary was used - allowing to access to its web application in localhost on the port 4200 with an automatic recompilation in case of code modification. Also, the format of the messages exchanged between the services was defined. It was key to standardize them for easy modification. This became difficult once variabilization and declarative configurations specific to Kubernetes were introduced. So, a popular development Kubernetes cluster running in a Linux virtual machine was deployed, Minikube. It is a single node cluster that is light enough for local use.

Under these conditions, the code present on the git allows with almost no modification to deploy on a Kubernetes Certified and Enterprise Grade cluster such as VMware TKG. It is necessary to first update the Docker images of the services uploaded on the Docker Hub platform. This abstraction is made possible thanks to the Kubernetes API, which is a real strength of this solution.


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organization
vmgouzy
Search by repository name...
Create Repository

vmgouzy / **imgform-handler-service** Updated a month ago	0	102	PUBLIC
vmgouzy / **kafka-test-ui-service** Updated a month ago	0	193	PUBLIC
vmgouzy / **claim-query-service** Updated a month ago	0	63	PUBLIC
vmgouzy / **claim-service** Updated a month ago	0	236	PUBLIC
vmgouzy / **image-analysis-service** Updated 2 months ago	0	45	PUBLIC

Figure 21 - Repositories in Docker Hub

Finally, the last phase is the deployment and testing phase. It was of extreme importance that the code would be easy to read so the next employee working on this application would understand every bit of it. In the end, the code was well-spaced out, correctly intended, and understandable. To achieve this goal, elements of explanation justifying the choices the team made were added. A lot of functions and code blocks are annotated in detail. As far as the deployment is concerned, it was done on the TKG platform. It was enough to simply restart from the command prompt and with the right kubeconfig file, automatically generated by TKG. All necessary commands were documented in the README of the Gitlab repository. Changes to deployment parameters (e.g., the keys to access MongoDB or the number of replicas of each service independently) can be made. The team started by deploying the application using a single replica for each service and then increased this number to three to check if the system was working properly under load. The Postman software allowed to send all kinds of HTTP requests in an automated way, to test the reactivity and the performance of the application, even in cases of peak load. The FormData POST is the most frequent type of HTTP request within the application as the main functionality is uploading images to be analysed.

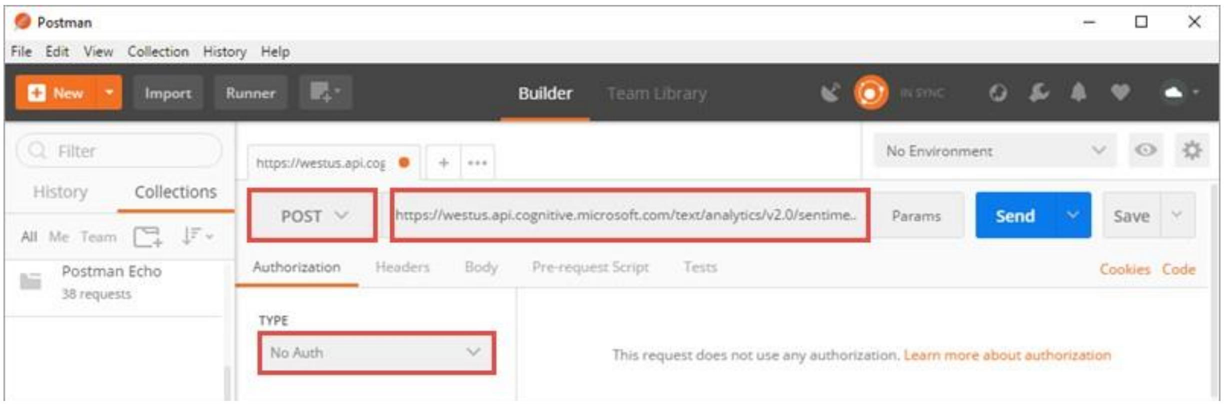


Figure 22 - Postman Interface showing an example of a POST Request

4.2 Differences with the initial specifications

Overall, the specifications have been respected. The application is functional and meets almost all the criteria defined beforehand. For reasons mentioned in the previous section, it is rather the timeline that has been less well managed, especially at the beginning. This allowed me to realize the importance of making regular updates to ensure the good progress of the project and not to allocate too much time in the provisional planning to the documentation and design phase as such, because these elements will probably be completed or even modified afterwards.

Although, as we have just said, we consider that the goal of the internship has been accomplished, there are plenty of improvements that could be done to achieve better results. Due to time constraints, no additional features could be implemented. Adding advanced log and metrics management; and the creation of a Helm chart for the deployment of the application are topics that deserve further research to optimize the application.

4.3 Overall Internship Experience

My employment period at VMware was very rewarding. VMware has a strong corporate culture and identity that is embodied in the EPIC2 values, which is an acronym for Execution, Passion, Integrity, Customers and Community. In my opinion, this culture is not just a marketing phrase but is strongly embedded in everything they do. This is outed by a relatively flat organizational structure where many employees are approachable and open to discussion. I have received tremendous support from my colleagues on both a personal and technical level. They have guided me and provided me with advice that helped me move forward, pushing myself out of my comfort zone.

Next to that, VMware is always seeking innovations to be part of a harmonious IT ecosystem. VMware solutions are the foundation for the digital player to make their solutions accessible in the most optimized way possible. The Office of the CTO is a global team dedicated to exploring and creating a future of disruptive technologies (Lavender, 2021). It focusses on near-term and long-term innovation initiatives that are aligned with VMware's business objectives, customer needs but ahead of product roadmaps. At VMware, we are also trusted technology advisors to our customers and partners, guiding them through their internal transformations. VMware sponsors academic research in universities globally, and sponsor both master’s and Ph.D. research internships, post-doctoral researchers, and affiliated faculty who focus on their long-term research agenda. The company also collaborate across the VMware ecosystem, co-innovating with customers and partners to deliver more value to their business. Furthermore, the innovator’s dilemma teaches us that any successful innovation over time may limit the options for future innovation as you become over-constrained by your prior success. At VMware, we practice both organic and inorganic innovation. The company focusses on strong internal innovation, inorganic innovation through acquisitions and business-model innovation in how we go to market and enable our customers to consume our technology in new ways — such as VMware Cloud (VMC) on AWS, Azure VMware Solution (AVS), Google Cloud VMware Engine, IBM Cloud, Oracle Cloud, and other cloud service providers globally. VMware also contributes to and utilize open-source technology when beneficial, and collaborate with academia, customers, and industry partners on co-innovation.

Overall, I am beyond grateful for the opportunity I got at VMware. As the project was technical and hands-on, I had a steep learning curve. I had a rather long theoretical learning phase, but it meant a lot to me that VMware cared about my personal enablement. I got exposed to an immense amount of unknown knowledge that was available to me and that could be useful for my project. When I needed to put that knowledge to use, I saw that different things got mixed up in my head. A key takeaway for the future would be to incorporate the documentation phase with the design phase. Thus, to start the design of a system as I research and experiment which is more of a trial-and-error approach. In my opinion, this could reduce the time before the beginning of the programming.

I also started coding by deploying the containers directly to my personal laptop with Docker. By doing this, when deploying the application to Kubernetes, I created unnecessary problems for myself. For example, I didn't understand that the front-end wasn't going to be on the same network as the back end, so I was joining the back-end services by their IP. This did not work afterwards, and I had to rework a few points in my program. For future projects, it would be important to develop directly considering the production architecture (maybe even emulate it like with Minikube for K8s).

5. LIMITATIONS

While building twelve-factor applications is a great starting point for developing applications that function in the cloud, it is necessary to explore beyond these factors to create cloud-native applications that can truly excel in the cloud environment (Hoffman, 2016).

Monitoring and auditing cloud applications are crucial aspects that are frequently disregarded, despite being among the most essential considerations for successful production deployments (Hoffman, 2016). While the cloud has simplified many processes, monitoring and telemetry remain challenging, and even more so than conventional enterprise application monitoring. The accuracy of telemetry can determine the success or failure of a cloud deployment. Application monitoring involves three primary categories of data: application performance monitoring (APM), domain-specific telemetry, and health and system logs. APM produces a stream of events that can be used to monitor application performance from external tools, and the data used for APM dashboards is typically generic and derived from multiple applications across several business lines. Domain-specific telemetry is a stream of events and data that is business-specific and frequently used for analytics and reporting. This event stream is usually fed into a big data system for analysis, warehousing, and forecasting. Finally, cloud providers should provide health and system logs, which are a stream of events such as application start, shutdown, scaling, web request tracing, and periodic health check results. When developing a monitoring strategy, it is crucial to consider the amount of data to be aggregated, the rate at which it will arrive, and the amount of storage required.

The original twelve factors do not address security, authentication, or authorization in any way (Hoffman, 2016). Security is crucial for both applications and cloud environments and should be considered from the outset rather than as an afterthought. A cloud-native application, by design, is inherently secure. It's important to keep in mind that the code, regardless of whether it's compiled or in its raw form, is transmitted through multiple data centres, executed within numerous containers, and accessed by many clients. Ideally, all cloud-native applications would employ role-based access control (RBAC) to secure all their endpoints. This means that every request for an application's resources must identify the user making the request and their assigned roles. The roles determine whether the calling client has the necessary permissions for the application to fulfil the request.

6. CONCLUSION

The focus of this internship report is to analyse key trends in software architecture both in research and real-world application, and to make informed predictions about the upcoming trends, obstacles, and goals. First, the drawbacks of a monolithic architecture are tackled, and to overcome these constraints, the software engineering community has been inclined towards cloud computing. Modern applications are increasingly being delivered and operated through cloud platforms, which are gaining widespread acceptance. The evolving infrastructural landscape has given rise to architectural styles that leverage the opportunities offered by cloud infrastructures. One such architectural style that has gained prominence in recent years and enables leveraging the benefits of cloud computing is the microservices architecture. By deploying applications in microservices, the software can regain evolvability, can become easier to maintain and scale, a clearer delegation of team responsibilities can be obtained, and developers can also make them fault tolerant.

VMware, an American software company that initially specialized in the virtualization of the data centre, transformed to adapt to the needs of today's IT world, where multi-cloud and modern applications are key pillars. As an SE intern, my main goal was to expand the possibilities of demonstrating cloud-native applications running on a VMware infrastructure and to integrate these features into an already existing demo application named Tito. The application automatised the return of rented vehicles. My job was to architecture the application to become cloud native and I added the functionality to automatise occurred damages to the car. To do this, I divided the work into different phases: a documentation, design, development, and deployment and testing phase. The two last phases have been very challenging as it was incredibly technical. Overall, I lived an experience rich in learning, both in an academic sense as on a personal level. For the first time, I had to design and develop a whole system, which made me gain a lot of autonomy. The different presentations I delivered, and the conferences I attended also made me a better communicator and developed my ability to synthesize. The few setbacks I experienced during this internship made me question myself and my way of working. I do feel I am now better equipped for my future professional career. I am extremely grateful to have lived this hands-on experience as I think it has been a perfect addition to the theoretical enablement, I have received at NOVA IMS. The practical internship experience and the academic learning complement each other perfectly. I also had the chance to work with evolving technologies such as micro-services and Kubernetes, which will become increasingly important in the future. An important discovery for me was that distributed architectures bring different issues to the surface such as data consistency or resistance to network partitioning. I would be curious to study these more depth in the future.

BIBLIOGRAPHY

Arora, C., Catlin, T., Forrest, W. K., & Vinter, L. (2020). *Three actions CEOs can take to get value from cloud computing.* McKinsey Global Publishing.

Atlantic. (2021). *What is VMware?* Retrieved from Atlantic: https://www.atlantic.net/dedicated-server-hosting/what-is-vmware/

Behara, S. (2019, June 27). *Making Your Microservices Resilient and Fault Tolerant*. Retrieved from DZone: https://dzone.com/articles/making-your-microservices-resilient-and-fault-tole-1

Bradshaw, S., Brazil, E., & Chodorow, K. (2019). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* Sebastopol: O'Reilly Inc.

Branas, R. (2014). *AngularJS Essentials.* Birmingham: Packt Publishing.

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2018). From monolithic to microservices: an experience report ffrom the banking domain. *IEEE Software*, 50-55.

Buckley, R. (2021, November 11). Leadership Talk: Intro to Tanzu. UK.

Castagna, R., & Brown, R. (2021, February). *Storage Virtualization*. Retrieved from TechTarget: https://searchstorage.techtarget.com/definition/storage-virtualization

Chen, R., Li, S., & Li, Z. (2017). From monolith to microservices: a dataflow-driven approach. *24th Asia-Pacific Software Engineering Conference (APSEC)*, 466-475.

Cloud Architecture Center, s. (2019, October 30). *Twelve-factor app development on Google Cloud.* Retrieved from Google Cloud: https://cloud.google.com/architecture/twelve-factor-app-development-on-gcp

De Lauretis, L. (2019). From Monolithic Architecture to Microservices Architecture. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 93-96.

Dragoni, N., Giallorenzo, S., Lafuenta, A. L., Mazzara, M., Montesi, F., & Mustafin, R. (2017). Microservices: yesterday, today and, tomorrow. *Present and ulterior software engineering*, 195-216.

Garg, N. (2013). *Apache Kafka.* Birmingham: Packt Publishing.

Garlan, D. (2000). Software Architecture: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering* (pp. 91-101). New York: Association for Computing Machinery.

Gelsinger, P. (2016, June 3). VMware Strategy: Value Proposition.

Gillis, T. (2019, July 11). *Avi Networks Now Part of VMware*. Retrieved from Blogs.VMware: https://blogs.vmware.com/networkvirtualization/2019/07/avi-networks-now-part-of-vmware.html/

Gos, K., & Zabierowski, W. (2020). The Comparison of Microservice and Monolithic Architecture. *IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS*, 150-153.

Haselböck, S., Weinreich, R., & Buchgeher, G. (2017). Decision Guidance Models for Microservices: Service Discovery and Fault Tolerance. *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems.*, 1-10.

Hoffman, K. (2016). *Beyond the Twelve-Factor App.* Sebastopol: O'Reilly Media.

Horowitz, B. (2016, July 28). *MRA, Part 5: Adapting the Twelve-Factor App for Microservices.* Retrieved from NGINX: https://www.nginx.com/blog/microservices-reference-architecture-nginx-twelve-factor-app/

Lavender, G. (2021, May 13). *A Culture of Innovation at VMware*. Retrieved from 3BL Media: https://www.3blmedia.com/news/culture-innovation-vmware

Lunden, I. (2018, November 6). *VMware Acquires Heptio, the Startup founded by 2 co-founders of Kubernetes.* Retrieved from TechCrunch: https://techcrunch.com/2018/11/06/vmware-acquires-heptio-the-startup-founded-by-2-co-founders-of-kubernetes/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAKNivR6IKppKwNJJJKQIEBq7nQap3MrOPj5UBwcBZAp2c4hhBKOEpvk7flFOGvzOz

Mall, R. (2018). *Fundamentals of Software Engineering.* Delhi: PHI Learning Private Limited.

Mardanov, A. (2014). *Express.js Guide.* Lean Publishing.

Miller, R. (2019, May 15). *VMware acquires Bitname to deliver packaged applications anywhere.* Retrieved from TechCrunch: https://techcrunch.com/2019/05/15/vmware-acquires-bitnami-to-deliver-packaged-applications-anywhere/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAKNivR6IKppKwNJJJKQIEBq7nQap3MrOPj5UBwcBZAp2c4hhBKOEpvk7flFOGvzOztvUyOd3l

Mortillaro, M. (2018, November 7). *Live at VMworld Europe 2018: An Update on VMware Strategy and Vision, Or VMware is Cool Again.* Retrieved from Kamshin: https://www.kamshin.com/2018/11/live-at-vmworld-europe-2018-an-update-on-vmware-strategy-and-vision-or-vmware-is-cool-again/

Nirgudkar, N., & Singh, P. (2017). The MEAN Stack. *International Research Journal of Engineering and Technology (IRJET)*, 3237-3239.

Nortal Cloud Team, s. (2017, January 18). *Accelerating time-to-market with microservices, containers and Kubernetes*. Retrieved from Nortal: https://nortal.com/blog/accelerating-time-to-market-with-microservices-containers-and-kubernetes/

Peltotalo, M. (2021). *A Case Study on Cloud Migration and Improvement of Twelve-Factor App.* Turku: University of Turku.

Pressman, R. (2005). *Software Engineering: A Practitioner's Approach.* New York: McGraw-Hill.

Pritchard, W. (2020). VMW VMware, Inc. (VMW) CEO Patrick Gelsinger on Citi's 2020 Global Technology Virtual Conference. [Recorded by P. Gelsinger].

Project Clarity, s. (n.d.). *Clarity Design System*. Retrieved from VMware.GitHub: https://vmware.github.io/clarity/documentation/v0.11/get-started

Sharma, C. K. (2020). Redundancy Optimization of a System under the the Fault-tolerant System. *Journal of Pure & Applied Science & Technology, 10(1)*, 1-5.

Singleton, A. (2016). The Economics of Microservices. *IEEE Cloud Computing, vol. 3, no. 5*, 16-20.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Emperical Investigation. *IEEE Cloud Computing, vol. 4, no. 5*, 22-32.

The Kubernetes Authors, s. (2021, July 23). *What is Kubernetes?* Retrieved from Kubernetes: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Villamizar, M., Garcés, O., Castro, H., Salamanca, L., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *10th Computing Colombian Conference (10CCC)*, 583-590.

VMware Tanzu, s. (2021, January 13). *Three Transformations Powering App Modernization*. Retrieved from Tanzu.VMware: https://tanzu.vmware.com/content/blog/three-transformations-powering-app-modernization

VMware. (2021, October 5). *VMware Tanzu Intro and Vision*. Retrieved from Modernapps.ninja: https://modernapps.ninja/introtótanzuportfolio_tp3617/docs/tanzuintrovision/

VMware. (2021). *About Us.* Retrieved from VMware: https://www.vmware.com/company.html

VMware. (2021). *Network Virtualization*. Retrieved from VMware: https://www.vmware.com/topics/glossary/content/network-virtualization

VMware. (2021). *What is Virtualization?* Retrieved from VMware: https://www.vmware.com/solutions/virtualization.html

Wiggins, A. (2017). *The Twelve-Factor App*. Retrieved from 12factor: https://12factor.net

Wikipedia. (2021, November 4). *VMware*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/VMware

Wikipedia. (2021, October 26). *Node.js*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Node.js

Wong, W. (2021, October 5). *VMware Pushes Multicloud Vision at VMworld 2021*. Retrieved from DataCenter Knowledge: https://www.datacenterknowledge.com/vmware/vmware-pushes-multicloud-vision-vmworld-2021

TechTarget. (2019, February 13). *The VMware CloudHealth acquisition extends cloud management.* Retrieved from CloudHealthTech: https://www.cloudhealthtech.com/news-media/vmware-cloudhealth-acquisition-extends-cloud-management

The Kubernetes Authors (2021). What is Kubernetes? Retrieved from Kubernetes.io: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

VMware. (2019, December). *VMware and Pivotal*. Retrieved from VMware: https://www.vmware.com/company/acquisitions/pivotal.html

Ross, A. (2019, November 5). *VMworld 2019 Europe: Democratising Kubernetes with VMware Tanzu.* Retrieved from Information-Age: https://www.information-age.com/vmworld-2019-europe-democratising-kubernetes-with-vmware-tanzu-123485989/

VMware. (n.d.). *VMware Timeline*. Retrieved from VMware: https://www.vmware.com/timeline.html

Thacker, M. (2021, November 1). *A New Chapter for VMware: Spin-Off from Dell Technologies Completed.* Retrieved from VMware News & Stories: https://news.vmware.com/releases/vmware-announces-completion-of-spin-off-from-dell-technologies

VMware. (2020, December 10). *Environmental, Social and Governance.* Retrieved from VMware: https://www.vmware.com/company/esg.html

VMware. (2021). *The Next Normal: Our 2030 Agenda.* Palo Alto: VMware Inc.